MASTER THESIS

Introducing Cooperativeness for Agrobotics: an Agent-Based Approach

Author:

Stef Bunte

S1981625

Supervisors University of Twente:

Dr. ir. M.R.K. Mes

Dr. Ir. E.A. Lalla

Supervisor Distribute:

Dr. Ir. B. Gerrits

August 2021

Management summary

In this thesis, we have developed a generic self-organizing control system for agrobotics. The design of such a self-organizing control system has multiple reasons, like a strong increase of scale, climate change, deterioration of arable land due to soil compaction, a decrease of labor, and stricter environmental laws. Due to many of these reasons, more robots are being developed to work more efficiently and reduce the needed man-hours. Many of these robots are pre-programmed standalone robots who are just doing one job. The system could be much more efficient if robots communicate and coordinate their tasks. Another important factor for change to robots is the decrease of labor. The decrease of labor causes that work cannot be done, is done inefficiently or inadequately.

We have made a Multi-Agent System (MAS) designed to control and communicate between the different entities in the system. We have designed the MAS with the Prometheus Methodology (Padgham & Winikoff, 2004). Every agent in a MAS can be seen as a piece of software situated somewhere in an environment and is capable of autonomous actions and decisions concerning a common goal (Wooldridge & Jennings, 1995).

All of this is summarized and covered in our main research question:

How would a generic self-organizing system look like for agrobotics applications characterized by simultaneous driving with pickup and delivery?

During the Prometheus design methodology, we go through three main design phases. In the first phase, the system specifications phase, we define the system goal and explain all the system's functionalities. In the second phase, the architectural design phase, we describe the agents and how they interact and communicate. We create a detailed overview of the agents in the detailed design phase. The MAS design consists of six agents divided over three levels, high-level, mid-level, and low-level. Each of the agents has its task.

- The Monitoring and Logging Agent (high-level)
- The Forecasting Agent (high-level)
- The Cooperative Agent (mid-level)
- The Vehicle Operating Agent (low-level)
- The Robot Status Agent (low-level)
- The Location Agent (low-level)

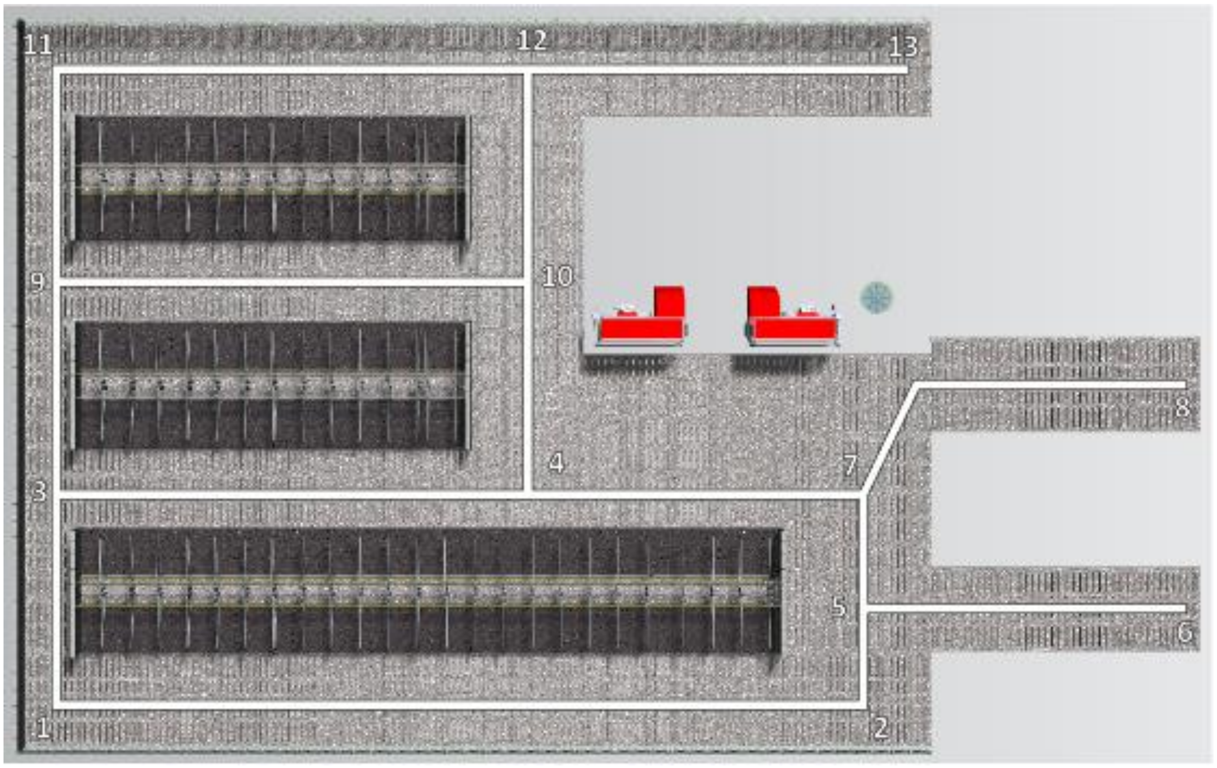
For genericity, we have designed the MAS, which is implemented with three different communication approaches. A central approach where the intelligence is located on the central point and the basic functionalities on the robot itself. A significant advantage of this system is that it can implement dynamic routing if continuous communication can be guaranteed. A second approach is a hybrid approach, where part of the intelligence is located on a central system and partly on the robot. In this approach, continuous communication is unnecessary, but the performance improves if the communication between the different robots and the central system improves. The advantage of this system is that if the robot cannot communicate with the central system, it can still make some intelligent decisions. The last approach is the decentral approach. In this approach, there is not a central system, but the individual robots have full intelligence. Because all robots have full intelligence, they can always make the best decision. The big downside of this approach is when decisions are made when the information of the other robots is not up to date. When this is the case, robots can conflict because the jobs are not appropriately aligned.

As a use case for testing the MAS, are we using a barn where two Lely Discovery 120 barn cleaning robots are working. Currently, the robots drive fixed routes according to a fixed schedule and have both robots their charger and dumping spot. Each robot also has its part of the barn where it is responsible for cleaning. When a robot gets a failure, the part of the barn where that robot is responsible is not cleaned until the failure is over. The failures can take up to hours, especially at night.

To cover the problems of this use case, we made three scenarios for improvement:

- Task handovers: in this scenario, a cleaning robot can clean areas usually cleaned by another cleaning robot. The robot only hands over its task when one of the cleaning robots fails, and that failure is known. This scenario has to be implemented at the same time as the following scenario.
- Shared facilities: in this scenario, it becomes possible to share lanes and share dumping spots. In that case, it becomes possible for cleaning robots to drive over lanes and dump their manure at the dumping spot, which usually is only used by another cleaning robot.
- Fully autonomous: in this scenario, the cleaning robot does not have a fixed schedule with preprogrammed routes anymore but only uses a heuristic to make and plan its routes.

When a cleaning robot takes over a route, a new route has to be determined. This routing problem can be categorized as a Capacitated Arc Routing Problem (CARP) (Essink, 2014). We have divided the barn into arcs and nodes. We have placed a node on every intersection and corner, and all those nodes are connected with arcs. Every arc represents a sector of the barn.



The heuristic determining the route the cleaning robot drives to that arc is based on the Nearest Neighbor Heuristic (NHH) (Winston, 2004). This heuristic is chosen because of its excellent fit with the problem and its simplicity. The Heuristic is first picking the arc that needs to be cleaned with the highest urgency. Which arc the robot cleans next is determined based on the last time of visit. The next step is to determine the cleaning robot which has the closest starting point to the sector. This cleaning robot is the cleaning robot that is going to take over the route. Then the heuristic starts to make the route from the starting point to the sector that needs cleaning the most urgent. This procedure starts by checking all the nodes that can be traveled to with one arc. The best node is selected by measuring the Euclidean distance from that node to the arc selected to be cleaned. This procedure continues until the arc that needs to be cleaned is reached. The heuristic uses the same procedure for going to the dump spot to dump its manure and going to the charger. Specific checks are built into the heuristic to prevent unnecessary driving or prevent the heuristic from getting stuck in a loop. There are also different strategies within this heuristic of picking the next sector to clean. The first strategy is the standard arc routing problem. The second strategy is prioritizing particular sectors that are assumed to be more critical to clean than others. In the third strategy are the larger sectors split up into multiple smaller arcs.

For the simulation study, we have defined four experiments. For comparing these experiments with each other, we are looking to the following KPIs. The first one is the average quantity of manure per area unit per time unit. The next one is the same as the first, only with a penalty for manure lying off a piece of land for a very long time. Thirdly, we make histograms of the number of area cleanings after a specific time and the number of area cleanings with a specific quantity. Furthermore, we look into the average number of failures, the average number of take-over routes, and the average response time.

The first experiment we have performed represents the current situation without any system failures. On average, there are 0.921 liters of manure per area unit and a penalty value of 3.208 in the current situation. To set a benchmark, we conducted the second experiment, where failures are activated. The KPIs in this experiment were somewhat higher, with 0.955 for the average quantity of manure per area unit and 3.422 for the penalty value. These numbers are 3.66% and 6.67% higher than the first experiment. 3.5 failures per six days cause this increase on average over two cleaning robots. From the histograms, we conclude that the interval of actions increases due to the failures. In the first experiment, there are no actions later than 4:15:00 and no greater than 2.75 liters. In the second experiment, there are on average 746.75 pick up actions and 592 pick up actions larger than 2.75 liters

In the third experiment, we are introducing the shared facilities and the task handovers. Additionally, we test the best communication configuration by running the experiment with different communication ranges between the cleaning robots and the central system. The result from this experiment is that the best communication configuration is to have at least 10 meters range from robot to robot communication combined with at least 10 meters range of robot to central system communication, where the robot can communicate with the central system at the chargers, automatic milking system (AMS), and at an additionally point at the side of the barn. In the second part of the experiment, we also added three different options for the heuristic. From these three strategies, is the strategy with prioritizing the best performing strategy. This strategy has more manure in the barn on average but reduced outliers in the histogram, minimizing the maximums.

The last experiment is the complete autonomous scenario. In this scenario, the cleaning robots do not have a fixed schedule with fixed routes anymore but determine all the routes on the heuristic. Over the whole barn, there was, on average more manure than in the current situation. But when looking at the results per sector, there are significant differences. The arcs with a dead-end perform much worse than the other arcs in the barn, which influences the results so much that they become worse than the current situation.

This research proposed a new generic MAS for agrobotics. This MAS is tested on a use case of barn cleaning robots of Lely. Due to assumptions and constraints, this research has its limitations. Therefore we recommend the following points for further research from a theoretical and a practical point of view:

Theoretical:

- The simulation model can be expanded in multiple ways. The first one is to model the behavior and visualizations of the cows into the model. If this modeling is done at the detailed level where the daily routines of the cows are modeled precisely, and the moments when and where they relieve their manure and urine, we can build a way more precise heatmap where the barn needs to be cleaned and when.
- The second expansion can be by adding other and different types of autonomous robots into the system. Robots like the feeding robot, AMS, or feed pusher can give much information about when and where it is more crowded in the barn. With this information, the cleaning robots can adapt their schedules to this. For example, the feeding robot gives a message that it starts feeding in 30 minutes. Then it is crowded at the feeding fence. The cleaning robots can adapt their schedule to clean beforehand and clean afterward not to interrupt the cows.
- When the first recommendation is implemented, it is also possible to expand the heuristic with a self-learning part. The heuristic has to keep up information about when and where it picks up how much manure. If enough of this data is logged, the heuristic can make a forecast when and where is laying how much manure. In this way, the cleaning robots can adapt their routes better, and the whole system's performance increases.
- To test the simulation model and the MAS in a different and larger barn where more than two cleaning robots work to see if the system is still performing and in this simulation study and as intended.
- Test the genericity of the MAS. The MAS should also be tested in a simulation study of another case, for example, H2Trac.
- Related to the heuristic, investigate the possibility to apply the heuristic to different cases outside the agriculture sector like robot vacuum cleaners and autonomous lawn movers.

Practical:

- First of all, the MAS should be implemented in the current system and tested in real life.
- The Lely should the Discovery 120 expand with a fill level measuring system. This information is beneficial and needed if the system is made more advanced.

Acknowledgments

This thesis which is laying in front of you, is the final work of my master's program Industrial Engineering and Management. During this thesis project, I have learned a lot above all the things I already had learned during my master's program and the two bachelor programs.

I have to say that finishing the thesis project was the biggest challenge I have faced during the nine years I have studied. During those years, I constantly have challenged myself to go higher with the motto: “Don’t worry, it will be fine.” And I think it is the phrase my parents, who always supported me during my study 100%, can’t hear any more from me.

This thesis project was a big project, which I knew at the beginning. Not only did I had to do tasks for my thesis, but I also needed to contribute to the consortium. These tasks were sometimes time-consuming, but I have learned a lot from them. Therefore I also want to thank all the people involved in the consortium for the things I have learned from them and the help and support they gave during this project, especially Jan Benders, the project leader from HAN Hogeschool and Koen Vermeu and Mauro Brenna from Lely, who offered a lot of helpful information insight for this project.

Secondly, I want to thank my colleagues at Distribute for all the support, help, and fun times, especially Berry and Robert. I look back with good memories of the time at Distribute.

Thirdly I want to thank my supervisors from the University of Twente, Martijn Mes, and Eduardo Lalla, for guidance and feedback on my project.

And finally, I would thank all my family, roommates, friends, and other people I forgot for their support and contribution in their own way. I hope I didn’t bore you too much with all the stories about cows, manure, cleaning robots, and tractors.

Stef Bunte
August 2021
Enschede

Table of Contents

List of figures

List of tables

1. Introduction

This document contains the report for the master thesis assignment. This master thesis project is part of the DurableCASE (Durable Cooperative Agrobotics Systems Engineering) project, which is a project in collaboration with the HAN university of applied science, Lely Industries, H2Trac, and many more parties.

This graduation assignment takes place at Distribute. This company is a spin-off company of the University of Twente owned by Ph.D. candidate B. Gerrits. This company is specialized in simulation studies of logistics systems with a focus on autonomous vehicles.

This report is structured as follows. In the first chapter, we provide the background for this research project, and we discuss the problem description with the problem context and the research questions. In Chapter 2, we present our findings of the literature review. In Chapter 3, the Prometheus methodology is worked out for this project. In Chapter 4, an extensive use case description is given for the simulation study, which is explained in Chapter 6. In Chapter 5 is the conceptual model described for this simulation study. The results of this simulation study are discussed in Chapter 7. In the last chapters, the conclusions and recommendations are given (Chapter 8 and Chapter 9).

1.1. Research motivation

The Dutch agriculture sector faces a strong increase of scale, climate change, deterioration of arable land due to soil compaction of heavy machinery, a decrease in labor, and stricter environmental laws. The reason for the strong increase of scale is that the profit margins on agriculture products are low, and that is why farmers have to expand their companies to be profitable. Therefore, they also need an extra pair of hands to complete all jobs on the farm. These changes result in the following problem: a decrease in labor. The decrease in labor is mainly due to the overall decreasing interest to work in the agriculture sector and that the low-wage jobs are not interesting for Dutch workers.

Due to climate change and stricter environmental laws, it becomes harder to yield the same crops as the years before. The last couple of years, for example, were very dry, and in some parts of the Netherlands, the harvest was utterly ruined. And due to the stricter laws, sprinkling dry fields and spraying herbicides against all kinds of insects and diseases are not always allowed anymore.

The trend in agriculture machinery was the last decade “just get bigger and bigger.” These machines have a high capacity and a high weight. Besides, there is still much-outdated machinery in operation. Old machines are often equipped with relatively small tires and are heavy, and together this has a high impact on soil compaction.

Like in every sector, there is a trend in the agriculture sector with increased robots, autonomous vehicles, and autonomous machinery. Those kinds of innovations can help solve a shortage of labor and parts a high level of soil compaction. Two companies developing and producing robots and autonomous machinery for the agriculture sector are Lely Industries and H2Trac. Lely is one of the biggest manufacturers of robotic solutions for dairy farms in the agriculture sector, and H2Trac is a young innovative company developing an electric tractor. The new concept tractor will be a small tractor that does not cause a high level of soil compaction.

1.2. Project introduction

The DurableCASE consists of two cases. The first is for small autonomous vehicles that bring the crops from the harvester to the end of the field to be loaded on larger road transport vehicles. These vehicles are also known as chaser bins.

The second case is about the Lely Discovery 120 barn cleaner (Figure 1-1). This case is the focus of this graduation project. The Lely Discovery 120 robot cleans the barn by collecting and sucking the manure into the machine. Further, the machine can spray water to get a cleaner result. A clean barn reduces the chance of injuries for the cows by slipping and reduces infections. An electrical motor and battery power the Discovery. Currently, the robots are driving pre-programmed routes in the barn and do not communicate which each other. Common failures that can occur are that the robots get stuck or they get an internal failure. In those cases, the robot has to be reset by a human. When this happens, it can take a while before it can be working again. Furthermore, the robots are currently not sharing facilities and lanes in the barn.

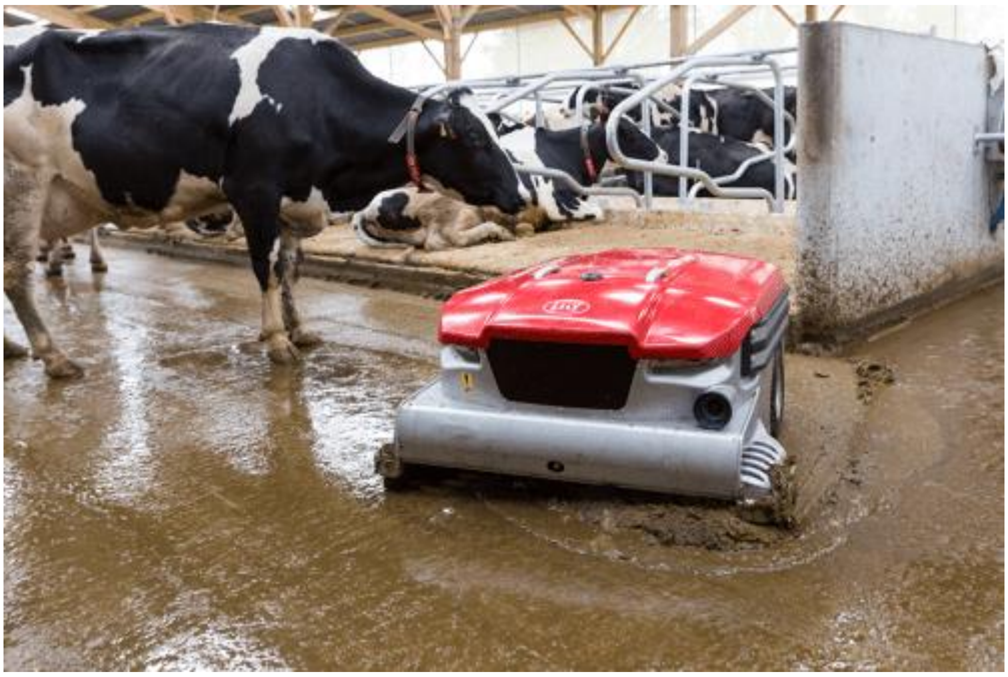


Figure 1-1: The Lely Discovery 120 barn cleaner

The DurableCASE project aims to create a generic and robust self-organizing control system that applies to various cases in the agriculture sector. The generic part of this project is that products have to be picked up or delivered in a specific place during driving at a specific time. For example, when driving along specific paths when cleaning large areas, harvesting crops, or following other vehicles with the master-slave concept. Applications that are suitable for such a self-organizing control system are, for example, seeding and planting crops, barn cleaning, or fertilizing fields. In some cases, the vehicle can even perform both to pick up and deliver products simultaneously. The self-organizing control system has to be cooperative with all the vehicles in the system and robust against all common problems.

This thesis project aims to create a generic self-organizing control system capable of assigning routes and jobs to robots based on the status of a dynamic environment and the status of other robots. This self-organizing control system must be designed so that it is applicable for multiple use cases with agrobotics. It should be possible to choose a central control, decentral control, or hybrid control approach for each case.

1.3. Problem context

In this section, the problem context is discussed. A root cause analysis is made to get to the source of each problem. This analysis is visualized with a problem cluster.

1.3.1. Lely case

The Lely Discovery barn cleaning robots are currently programmed so that every robot can drive a couple of pre-set routes. These routes start and end at the charging station. The manure dumping at the dumping spots is also programmed in these routes as well. When multiple robots work in the same area, they all have their own charger and routes and do not share any lanes or barn facilities. The robots cannot communicate with each other, and they only log the battery status and the driven routes to a cloud environment. Lely wants to introduce shared places and task handover in the system to improve the system so that the robots can be more efficient.

When introducing shared places, the robots may block each other's paths. One of the biggest problems is that the robots get stuck, lose their location, or fail and do their jobs anymore. In that case, the farmer gets a notification of the error. To make it visible how all the problems are connected, we have made a problem cluster in Figure 1-2. In this way, it is also known what the root causes are of the main problem. Which problem and which root cause is more important than others is determined by Lely self and are colored green in Figure 1-2 and are described below.

Most of the causes of the problems are related to congestion at shared places. Shared places are the dumping spots, charging spots, and lanes that can be assigned to multiple robots. Furthermore, communication problems partly cause congestion. The communication problem is that the robots do not communicate with each other and do not know where other robots are and their status.

Another problem is when the distribution of manure is different from usual. This difference can happen due to weather changes, and cows are more outside in the field or more inside the barn than usual. The diet and the lactation state of the cows impact how much manure they produce.

A root cause that humans cause is the manual blocks for specific routes temporarily. The robot owner can choose to skip a couple of routes temporarily. A reason for this could be that part of the barn changes in the waiting area for milking. This event also interrupts the system quite heavily because areas have to be cleaned at a certain point or cannot be cleaned for a while.

The last root cause is related to problems with the pre-planned routes. Those routes have to anticipate the daily schedules and routines of the barn, like feeding times and milking times. The routes should also anticipate certain areas of the barn to become dirty faster than others. When the robot does not anticipate correctly, there is a higher chance of getting stuck in congestion with other robots, and the barn is not clean enough overall.

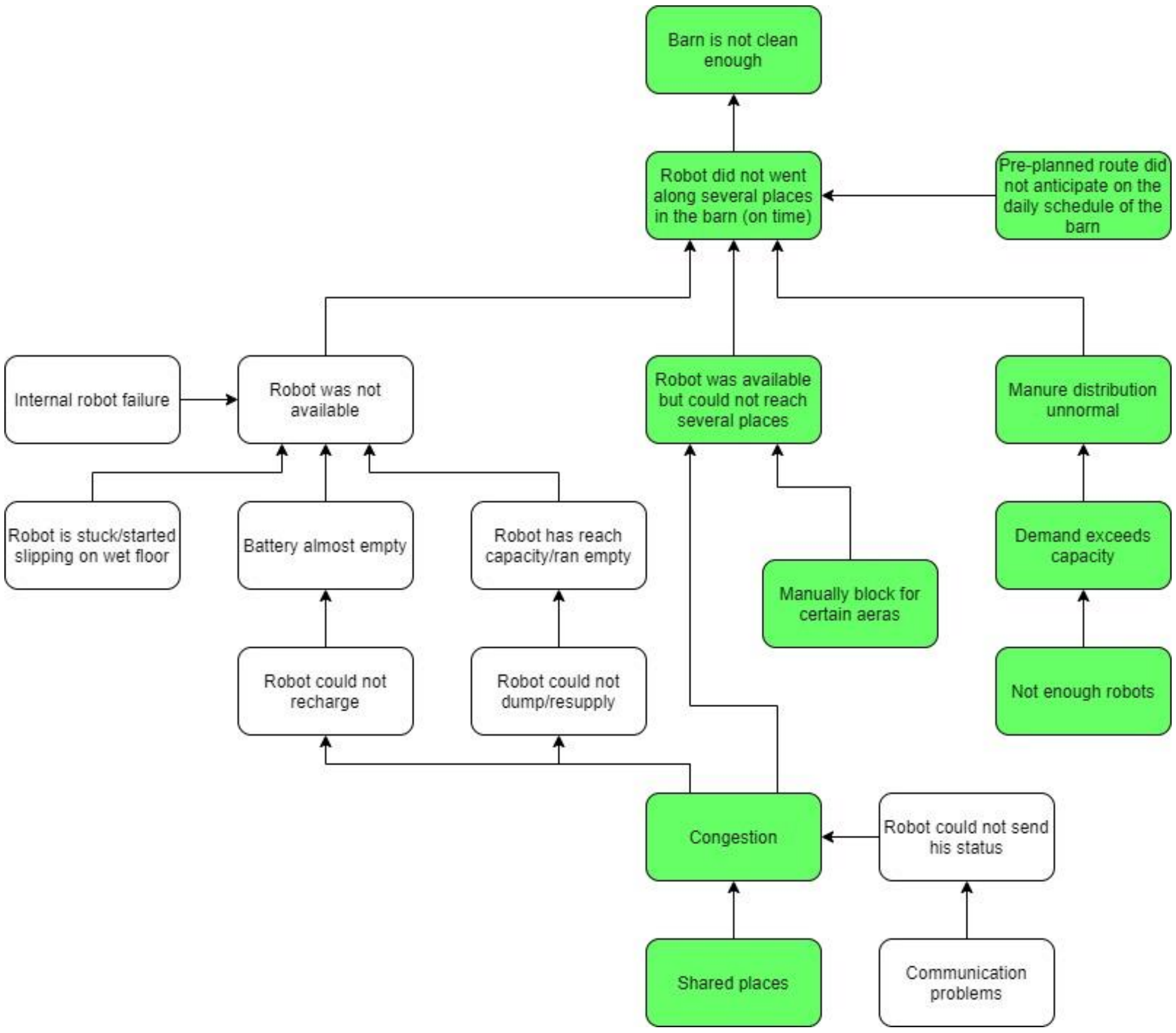

Barn is not clean enough
Robot did not went along several places in the barn (on time)
Pre-planned route did not anticipate on the daily schedule of the barn
Internal robot failure
Robot was not available
Robot was available but could not reach several places
Manure distribution unnormal
Robot is stuck/started slipping on wet floor
Battery almost empty
Robot has reach capacity/ran empty
Manually block for certain aeras
Demand exceeds capacity
Robot could not recharge
Robot could not dump/resupply
Not enough robots
Congestion
Robot could not send his status
Shared places
Communication problems

Figure 1-2: Problem cluster Lely case

1.4. The core problem

It became apparent in the last section that most root causes can be categorized in the category routing problems. This category covers the problems of not driving the best route, not starting the route on time, and lacking the fleet's capacity.

From the project management of DurableCASE, it is required that the robots must interact with each other to create a robust and cooperative system that reduces the number of problems in the system. The interaction is needed for two reasons: firstly, in some cases, the pick-up and delivery spots can be dynamic in the environment with a certain level of uncertainty. The second reason is that the vehicles have to communicate and help each other when one has a problem. To improve the system, the robots in the system should know the current status and location of the other robot and how to adapt their current actions to that new information. In that way, the routing and planning of the robots should improve. This situation is perfect for the implementation of a self-organizing control system.

So, the core problem is not knowing how the autonomous vehicles should make decisions, how to communicate, and not knowing where to go knowing the position and status of the other (autonomous) vehicle(s) and the status of the environment.

1.5. Assignment

The assignment is to create a self-organizing control system that can act autonomously, cover the routing and communication between robots in the system, and, most importantly, apply the system to multiple cases concerning agrobotics. The project consortium wants to design a generic Multi-Agent System (MAS) to create a self-organizing control system. The DurableCASE project consists of two cases: Lely and one from H2Trac, with multiple scenarios. The scenarios of the Lely case are tested with a simulation study to see how the system reacts and which strategy performs the best. With this, they want to eliminate/reduce the most critical problems.

1.6. Data collection and management

DurableCASE is a project where many companies and universities are involved. To handle the information between all the parties, the group leader, HAN university of applied science, has set up multiple meetings where vital issues are discussed with parties that have knowledge about the problem.

A simulation study acquires the essential data and information on which the conclusions and recommendations are based. This simulation study is tested on how the vehicles should move and interact with each other. This information comes from Lely. They provide log files of the routes of a real-life situation in a barn with two robots. These log files consist of statistics about the start and end times of the routes and all the alarms the robot gives.

1.7. The problem approach

To solve the core problems, an intelligent and cooperative routing system has to be implemented. This routing system has to be controlled by a system that can make decisions on its own. Therefore, the first step is to do literature research on self-organizing control systems and design such a system. The second part of the literature study is on different routing problems that can or cannot be used in this project. In both cases, it is possible to divide the environment into arcs and nodes. So, it is possible to tackle the routing problem with an arc routing problem.

Also, a literature study will be done on this topic to gain more information on the background, self-organizing control systems, how to design those systems, state of the art in agrobotics, and the current developments of autonomous vehicles in the agriculture sector. We know what is possible with that information, what exists, and what implementations we can apply to the system. The second step is to design a self-organizing control system systematically with the usage of a methodology.

When the methodology is completed, we can start with the conceptual model for the simulation model and build this simulation model. For this, we first create the current situation in the model as well as possible. By doing so, we can check how the improved situation differs from the current situation. With the current situation in the model, we can easily adapt the model to our designed situation. Different kinds of multi-agent systems with different scenarios are modeled and tested.

Because the environment where the machines are working is changing continuously and cannot fully predict how this changes, it is tough to get the optimal solution. This project aims not to get the optimal solution but to get a reasonably good solution that improves the current.

In the end, we evaluate the results of the simulation study. Based on these results, we draw up the conclusions and give our recommendations for the companies and suggestions for further research.

1.8. The research problem

The main goal of the DurableCASE project is to create a robust automated planning and control system based on MAS technology for Simultaneous Driving with Pickup and Delivery (SD PD) by means of cooperative vehicles in a self-organizing environment in such a way that it yields a cost-effective solution.

The big difference with standard autonomous vehicles in logistics is that many autonomous vehicles in the agriculture sector pick up or deliver products while driving at a specific place, where most autonomous vehicles bring the product from point A to point B.

As said before, the focus is on a self-organizing system in the current situation and the problems that such a system can solve. So, problems that are caused by technical problems or machine design are out of scope. Also, ground and weather conditions cannot be changed using a self-organizing system, so this is also out of the project's scope. What is in or out of the project's scope is discussed in more detail in Chapter 4 and Chapter 5.

A self-organizing system is designed in this thesis project based on the Lely case. We design three different self-organizing systems to make them as generic as possible: a central control approach, a decentral control approach, and a hybrid control approach. The Lely case is used in the simulation to validate the self-organizing system. In the use cases, we research the cases when the robots use shared places and what to do when a robot is temporary out of service. Together with Lely, we made assumptions about the system for the simulation study. We discuss these assumptions later in this report. Furthermore, some functionalities were made that were either necessary or nice to have in the new and improved situations. These are also discussed later on in this project.

1.9. Research questions

Like any project, this project has one main research question and multiple sub-research questions to answer the main research question. The main research question is:

How would a generic self-organizing system look like for agrobotics applications characterized by simultaneous driving with pickup and delivery?

The first set of sub-questions is answered with the literature research. When these questions are answered, we know what we can do with self-organizing systems in combination with agrobotics, in which cases it is implementable and what kind of research there is already done on similar cases.

What is a self-organizing system, and how can it be designed for agrobotics?

- *What is a self-organizing system, and where can it be implemented?*
- *What methodology can be used to design a self-organizing system?*
- *What are related vehicle routing problems, and how can these be used in this case?*
- *What is the state of the art of autonomous robots in the agriculture sector?*
- *What is related work available concerning self-organizing systems in the agriculture sector?*

The second set of sub-questions is answered with the methodology that is used to design the self-organizing system. By the end of Chapter 3, we know how a generic and robust self-organizing system for agrobotics looks.

How would a generic self-organizing system look like for agrobotics with simultaneous driving with pickup and delivery?

- *Which cases in the agriculture sector should the self-organizing system control cover?*

- *How would a self-organizing control system look like for the use case?*
- *What types of self-organizing control system approaches are possible in the agriculture sector?*

Chapter 4, Chapter 5, and Chapter 6 answer the third set of sub-questions. These questions are focusing on the conceptual model and the simulation study.

How would a simulation model look like for a self-organizing system for barn cleaning robots?

- *What scenarios are relevant for the barn cleaning robots to research?*
- *What data is needed to model the barn cleaning robots?*
- *What conceptual model would be suitable for this simulation study?*
- *How to implement the conceptual model into the simulation?*

In the last phase, we ask how the different approaches perform and what the improvements are in the system if a self-organizing system is implemented in the use case.

How is a self-organizing control system for barn cleaning robots performing?

- *What are the differences in performance between the different approaches?*
- *Which performance improvement is expected of the barn's cleanliness in the Lely case if the robots communicate with each other and adapt their routes based on the received information?*
- *What are the performances of the system when the environment is changing?*
- *What are the performances of the system when the robot is chancing?*

These questions answer the work that is done through the following chapters. At the end of each chapter, they are summarized and concluded.

1.10. Deliverables

This thesis project results in the following deliverables:

- A generic self-organizing system designed for cooperative robotics in the agriculture sector with a central, decentralized, and hybrid approach is motivated by the Lely case.
- A simulation model and study of the self-organizing control system design for the Lely Discovery 120 validated the generic self-organizing control system.
- A written report with all the results, conclusions, and recommendations.

These products are made available for the whole consortium of the DurableCASE project.

2. Literature review

This chapter describes the literature review for this project and answers the first sub-research question; *What is a self-organizing system, and how can it be designed for agrobotics?* In this project, there will be a self-organizing control system developed. Therefore, we introduce a self-organizing system and different methodologies to design such systems in this literature review. Secondly, as mentioned in the chapter before, the routing problem we face in this project is a so-called arc routing problem. Therefore we also address the different arc routing problems and how they fit this project. After that, we are going to elaborate more on MAS. Finally, we discuss the state of the art of robotic and automatic solutions in the agriculture sector.

2.1. Self-organizing systems

In this part, we are going to discuss Multi-Agent Systems (MAS). MAS is the self-organizing system that is implemented in this project. The choice for MAS is made by the consortium. First, we discuss what a MAS is. Secondly, the applications of a MAS. And at last, the current state of MAS in the agriculture sector.

2.1.1. Definition

A system is a self-organizing system when it can make decisions on its own, overcome failures, and function without big interferences. Also, a self-organizing system should be adaptive, needs only minimal data to function, and strives for a common goal (Bartholdi et al., 2010).

2.1.2. Multi-agent system

A multi-agent system is a software system that consists of multiple pieces of intelligent software, the agents, with each having their own tasks. An agent can be described as follows:

'An agent is a computer system situated in some environment, and that is capable of autonomous action in this environment to meet its design objectives' (Wooldridge & Jennings, 1995).

This citation is a short definition. We will now specify all the characteristics of an agent. The first and most important characteristic is that every agent is *autonomous* and can always do their essential task independently without interacting with other agents. This characteristic makes a MAS *robust* against failures and errors. Next to that, agents always have multiple ways of succeeding in their goals, making them *flexible*. Also, agents are *proactive*, which means that they can take the initiative without getting an order. To reach their goal, they often have to interact with other agents and be *social*.

Another essential characteristic is that the agents are *situated* in some environment. Of course, every piece of software is situated in some environment, but the environment of agents can be described as dynamic, unpredictable, and unreliable. These characteristics mean that the environment can change very quickly, and that the agent cannot rely on the fact that the environment stays the same while completing a task or achieving a goal. Because of this changing environment, the agents have to *react* promptly to changes in the environment.

Following Zambonelli et al. (2003), there are two types of multi-agent systems; distributed problem-solving systems and open systems. In distributed problem-solving systems, the agents are designed to work toward the same given goal cooperatively. In open systems, this is not the case. These cooperative systems consist of multiple agents that do not have a common goal and are designed by multiple parties. Moreover, agents may enter and leave the system, so the system composition is dynamic.

2.1.3. Applications
Multi-agent systems can apply in very diverse environments and diverse applications. In this project, we are working on physical robots in the agriculture sector. But it can also be applied in pieces of software that work together to achieve a common goal. Padgham & Winikoff (2004) use an example in their methodology of a multi-agent system that handles an online book store. In this example, multiple agents have their own tasks and responsibilities, such as taking the customer orders, keeping track of inventory, and handling backorders.

2.1.4. Multi-agent systems in the agriculture sector
Janani et al., (2016) have researched what the best strategy is for headland management. The case they use is with multiple robots of the same kind that perform the same task (plowing) in lanes directly next to each other. They compared two strategies; First In First Out (FIFO) and Last In First Out (LIFO). The headland management must be done in one of those two strategies because most of the operations performed in the fields cannot be performed simultaneously due to machine constraints. The idea with the FIFO strategy is that when a robot finishes a lane, it drives over the headland to the next lane, where it has to work. The other robots follow the first robot in the sequence of finishing the job. When the last robot finishes with its lane, the first robot starts again.

With the LIFO strategy, the robots make a line on the headland when they finish their lane. When the last robot is finished, it can immediately start with the next lane. The robot which had finished first starts last with a new lane. This strategy requires much less headland because the robots do not have to drive around each other (Zambonelli et al., 2003).

Chevalier et al., (2015) have introduced a decentralized multi-agent system for precision agriculture. They introduce a combination of Unmanned Ground Vehicles (UGVs) with Unmanned Aerial Vehicles (UAV). The UAV flies continuously above the group of UGVs and functions as an extra sensor for the UGVs.

2.2. Design methodologies
To get structure in the design process of the self-organizing system, we use a methodology where we can follow the different design phases and steps to get a successful result. In this section, we address multiple options that can be used.

2.2.1. AOM, ADPEPT, and DESIRE
First, we are going to discuss the paper of Shehory & Sturm (2001). They are comparing three different methodologies on 15 different criteria; this is summarized in Table 2-1. The three methodologies are the Agent-Oriented Methodology (AOM), the Advanced Decision Environment for Process Tasks (ADEPT), and the Design and Specification of Interacting Reasoning (DESIRE).

AOM is a methodology that focuses on the modeling aspect of agent-based systems. AOM uses different models through the designing process of the agents for both the analysis and design phases. When following the methodology guidelines, the analysis gives a set of roles for the system. For each role, we need to define the permissions, responsibilities, and protocols. Also, an interaction model is made with all the information on the interactions between the roles.

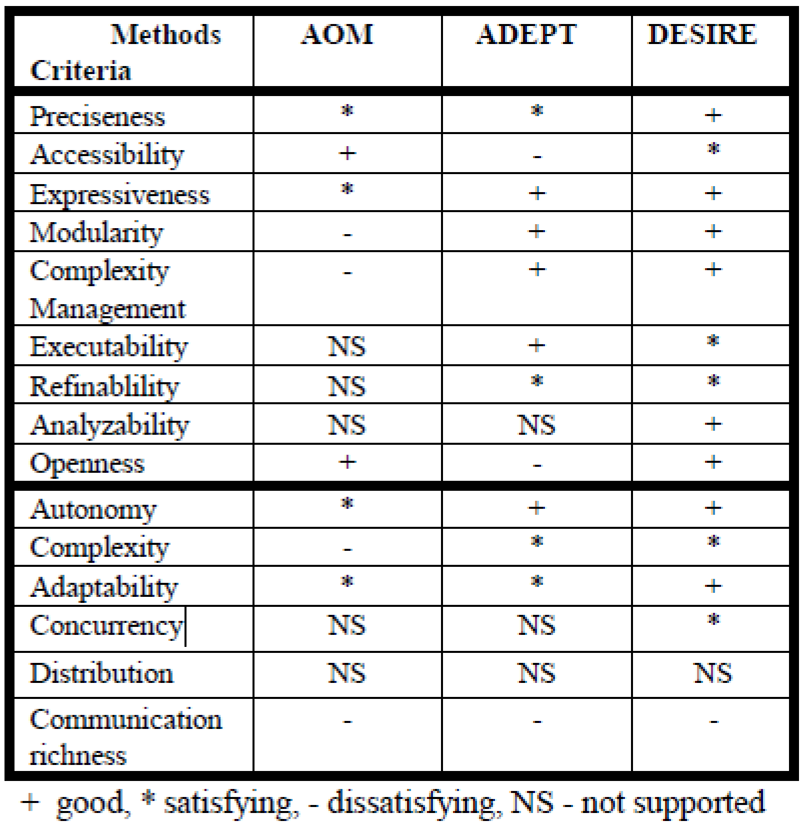
AOM is easy to understand due to its straightforward models, and it is an open modeling technique. A downside is that the methodology is only applicable to small and medium-sized systems.

In contrast to AOM, the ADEPT methodology does include implementation in the methodology. This methodology gives a set of models and languages to achieve its purpose. The ADEPT methodology uses building blocks that consist of an agent, a set of tasks, and sub-agencies. The communication between agents in the ADEPT methodology is defined with protocols based on the speech-act theory. Much of the methodology is documented in a programming language.

The DESIRE framework covers the whole design and implementation of a multi-agent system, just like the ADEPT methodology. The DESIRE framework uses the following models: (1) task (de-)composition, (2) information exchange, (3) sequencing of (sub-) tasks, (4) subtask delegation, and (5) knowledge structure. The task (de-)composition visualizes the input, output, and relations hierarchy between the tasks. The needed information to complete all the tasks is covered in the information exchange model.

A benefit of this framework is that it is an open system and does not use a specific programming language or architecture. The downsides of this framework are that the messages between components are not defined, and the framework is quite challenging to implement.

Table 2-1: Methodology evaluation (Shehory & Sturm, 2001)



Methods / Criteria	AOM	ADEPT	DESIRE
Preciseness	*	*	+
Accessibility	+	-	*
Expressiveness	*	+	+
Modularity	-	+	+
Complexity Management	-	+	+
Executability	NS	+	*
Refinablility	NS	*	*
Analyzability	NS	NS	+
Openness	+	-	+
Autonomy	*	+	+
Complexity	-	*	*
Adaptability	*	*	+
Concurrency	NS	NS	*
Distribution	NS	NS	NS
Communication richness	-	-	-

+ good, * satisfying, - dissatisfying, NS - not supported

2.2.2. Gaia methodology

The Gaia methodology (Zambonelli et al., 2003) is an extension of AOM (Shehory & Sturm, 2001). The Gaia methodology consists of five phases: The collection of requirements, Analysis, Architectural Design, Detailed Design, and Implementation (Figure 2-1).

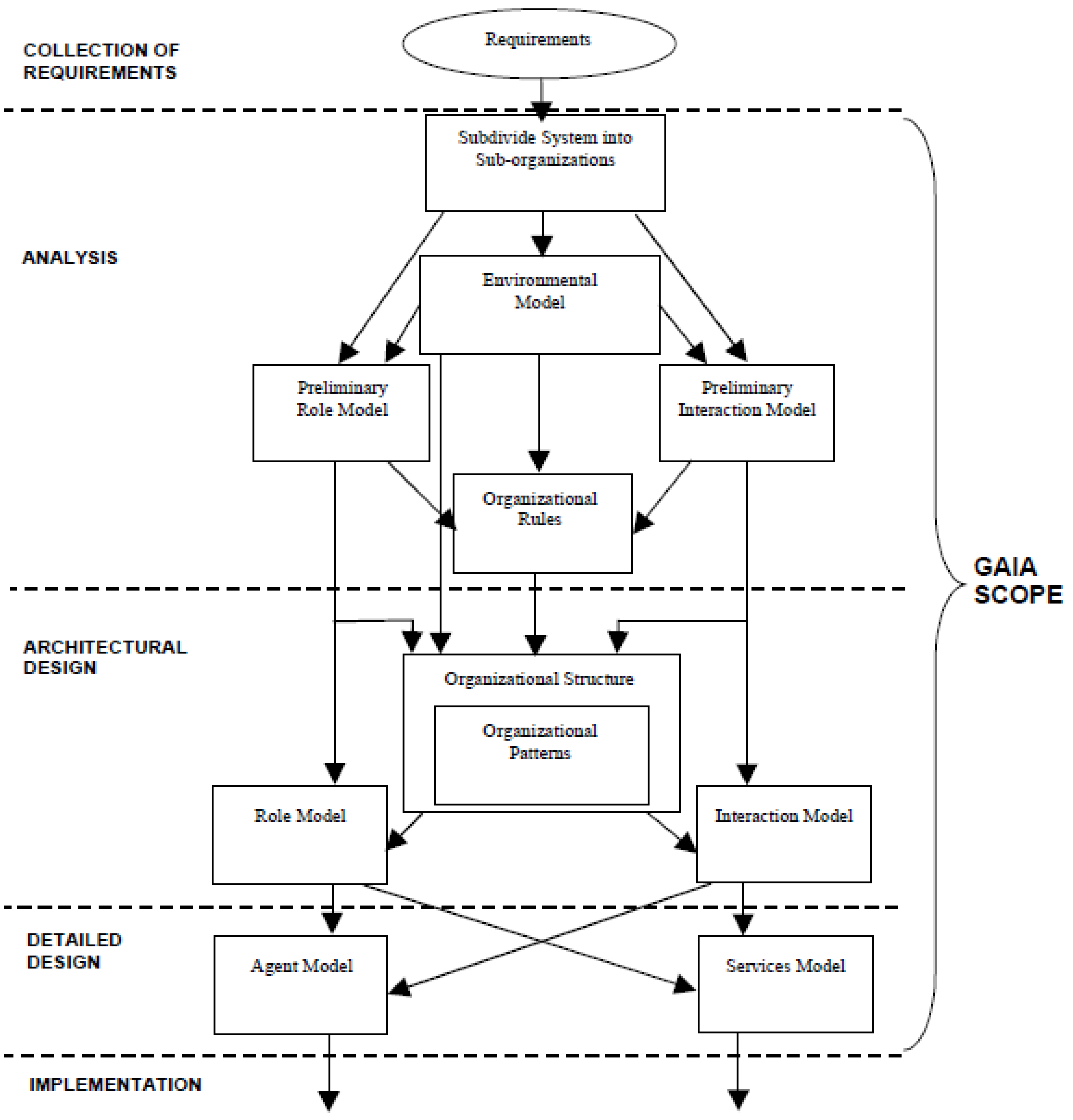


Figure 2-1: Gaia methodology (Zambonelli et al., 2003)

When the requirements are known, the analysis can start. In this phase, multiple models are identified:

- The goals of the organizations constitute the overall system and their expected global behavior
- The environmental model
- The preliminary roles model
- The preliminary interaction model
- The rules that the organization should respect and enforce in its global behavior

The models and information gained from the analysis phase are used as input for the next phase, where the architectural design gets its shape. In this phase, the design gets more structure, and the control regime is defined. Next to that, the preliminary models from the last phase are completed.

The information from the architectural design phase is used as input information for the next phase, where the design gets its details. In this phase, the agents are defined and modeled. Definition and modeling are done based on the interaction between the roles. Most of the interactions are related or similar. For convenience and efficiency, these are grouped in the same class. To realize the agent's roles and properties, a service model is defined at the end.

Next to that, the Gaia methodology is not committed to particular modeling techniques. So it is open for the methodology's user to use different modeling techniques for making the models in the different phases. That the methodology is open for users can also be seen back at the implantation phase. The end design of the methodology is not directly implantable programming software.

2.2.3. Prometheus methodology

The Prometheus methodology is a methodology that is also specially designed for designing multi-agent systems. This methodology leads the designer of the system systematically through the project. The whole methodology is split up into three main phases (Figure 2-2). The system specifications, architectural, and detailed design phases are connected (Padgham & Winikoff, 2004). In this part of the report, we will go through them step by step.

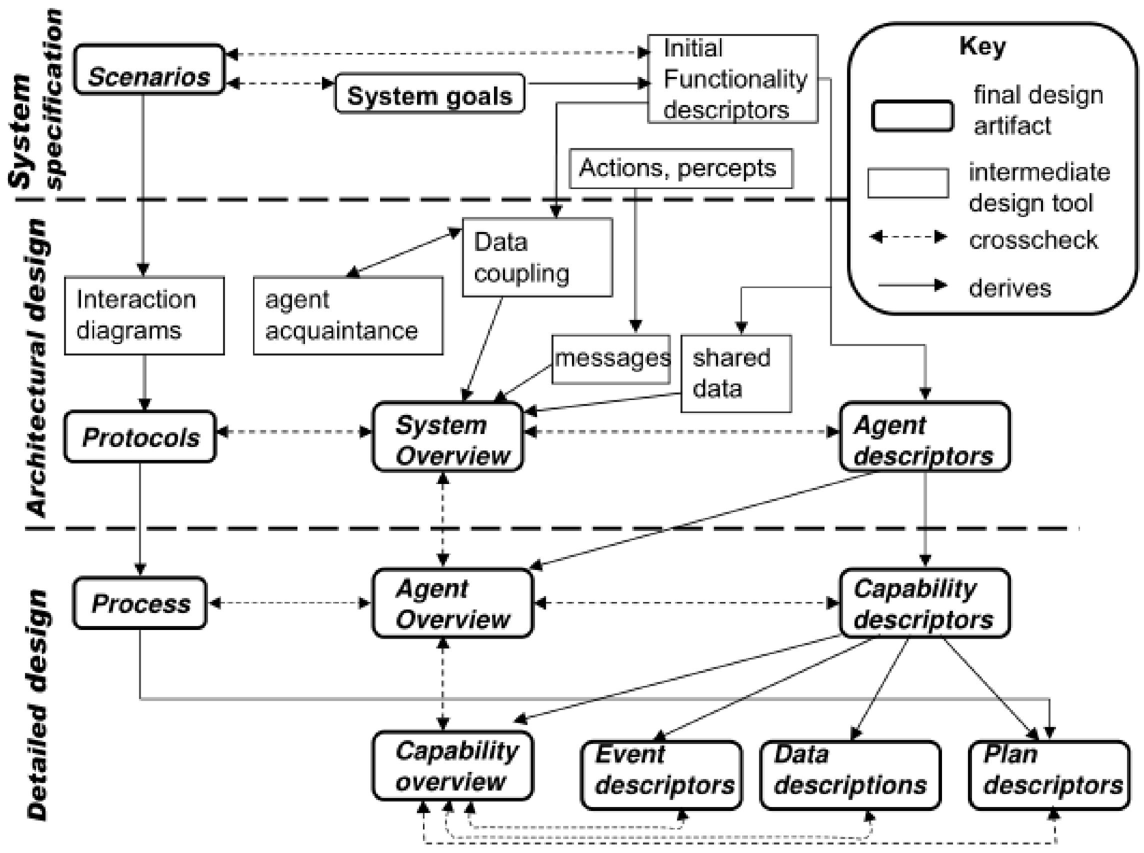


Figure 2-2: Prometheus methodology

Systems specification phase

As the name already implies, in the system specification phase, the system's specifications are determined. There are multiple steps in this phase: the first one is identifying the system goals. The system goals are listed with the system's overall goal and all the elements to reach that primary goal. This primary goal is split down into many sub-goals.

The next step is to make a list of all scenarios that could happen in the system. It is possible to identify a list with percepts and a list with actions based on that list. Percepts are signals that a system can receive from sensors. For example, maximum capacity reached. Actions are all the tasks a system can perform based on the percepts it has received. For example, dump goods. This action can be an action that can follow from the percept: maximum capacity reached.

After that, all functionalities need to be determined. A functionality can be seen as a chunk of behavior. This chunk of behavior includes related goals, percepts, and actions. It is essential to make a functionality for every simple task that the system has to perform. If it is impossible to describe a functionality in one or two sentences, it is too complex and needs to be split into multiple functionalities.

The next step is to define some use case scenarios to illustrate the system's operation. These cases consist of all the standard actions an agent can perform, which can also be problems that can occur.

Third, the basics of the system are identified. And finally, the percepts and actions of the agent have to be specified. These are the input and output for this phase.

Architectural design phase

The second phase is the architectural design phase. In this phase, it is decided which types of agents are used to interact in the application. This phase uses the output of the systems specification phase as input and creates input for the detailed design phase.

The first step is making a data coupling diagram to decide how the agents will look. This diagram visualizes how the data flows between the functionalities. When this is done correctly, groups with data and functionalities are automatically formed. Those groups are good options to deploy in a single agent. It must be considered that it is necessary to have multiple agents of the same type in the system. In the case of functionalities living on robots moving around in the system's environment, the system has multiple agents of the same type. The number of agents must not be too high or too low. The correct number of agents is essential to balance the complexity of the agents and the interactions. When the number of agents is too high, the system becomes too complex. When the number of agents increases, the number of interactions needed between the agents also increases. When the number of agents is too low, the agents themselves become too complex. Furthermore, a good balance between the number of agents and interactions positively influences the overall robustness of a system.

The second step is to evaluate the design that has been made. This evaluation is done by calculating the link density between agents and making an agent coupling diagram. An agent coupling diagram visualizes which agents are interacting with each other. This visualization is done by drawing lines between the agents. The link density can be calculated by taking all links in this system and divide this by the total potential links. This calculation has to include the possibility of having multiple similar agents. The closer the link density gets to one, the higher the chance of getting a run-time bottleneck. A design with a lower link density performs better than high link density, and therefore are these designs more preferable. Lastly, we summarize all information per agent in the agent descriptors. These also contain information about when the agent is active, the actions he has to take when he wakes up and when it is demising.

The next step is to specify the interaction between the agents. These interactions capture the dynamic aspects of the system. First, the interaction diagrams are developed for every scenario. These diagrams show from top to down which actions and percepts are activated, which agent responds to it, and how messages are sent between agents. From these diagrams, it is possible to make a protocol diagram. Protocol diagrams and interaction diagrams are similar, but the protocol diagrams describe smaller and more intricate parts of the interaction diagrams. In the protocol diagrams are also different options and alternatives visualized. At the end of this step, we summarize the protocols and messages in descriptors with all relevant information.

The last part of this phase is about finalizing the architectural design phase—the finalizing starts with setting the agents' boundaries. The next step is making descriptors of all the percepts and actions and defining shared data objects. At the end of this phase, a system overview diagram is made. This diagram visualizes how all percepts, actions, agents, protocols, data, and messages are linked. In that way, the whole system can be checked if everything is linked correctly and described in the design.

Detailed design phase

The third and last phase is the detailed design phase. As the name already suggests, this phase is about the details of the agents and system. And just like the previous phase, this phase uses the outcome of the previous phase as input.

The first step in this phase is to set all the capabilities of every agent. Which capability every agent has is based on the functionalities. Every capability has its own goal or multiple goals.

The next step is to make agent overview diagrams. These diagrams show the relationships between the capabilities and give a top view of the internal system of the agents. The input information is the goals of each capability and the information from the protocols of the previous phase. The diagram has much in common with the system overview diagram, but instead of the agents, it shows the capabilities.

After this, the process specifications are defined. During the previous phases, the processes were specified by scenarios. These scenarios were further specified in the last phase with interaction diagrams and protocols. In this phase, the processes are more specified and structured. This structuring can be done by multiple variants of UML (unified modeling languages). These so-called UML diagrams must clarify in which sequence steps are done in the processes. The capabilities and processes are summarized in descriptors again after the process specifications are made.

The next step is to make for every capability a capability diagram. These diagrams, which are often quite complex with incoming and outgoing items, are split up into smaller diagrams containing only that specific capability's activities.

The following step is to define all the tasks and plans to the lowest level of detail. Every capability needs to have at least one plan on how to respond to an incoming message. These plans use programming control structures and binary statements. Important here is that everything is covered very precisely for every case that can happen. These plans are also summarized in descriptors and need to carry all information to understand the capability overview diagram and everything needed for the implementation phase.

The final step in this phase is to check if everything is complete and connected correctly with each other.

2.2.4. Comparison

In the sections above are the different methodologies discussed. From the project point of view, the Prometheus methodology is the best suited. This choice is based primarily on the fact that Distribute already has experience with this methodology and that the models used are good visualizations. The other methodologies use models that contain fewer visualizations by figures but are more written in a kind of programming language. Because the DurableCASE has many partners who have to understand the design and do not all have the knowledge to read these kinds of languages, we have chosen a more visual approach.

2.3. Routing problems

Because routing is an essential part of this project, we look into different routing problems similar to ours. As said before, we are looking at an arc routing problem. Because there are different arc routing problems, we first give an overview of the different arc routing problems to find the best match with our routing problem. We explain this best match more in-depth and how it can contribute to our problem. The next step is to introduce a couple of heuristics that can solve arc routing problems and could be helpful to solve our routing problem. At last, we give a small introduction about the heuristic used by the Lely Discovery 120 to localize itself in the barn and drive the routes.

2.3.1. Arc routing

Arc routing problems are vehicle routing problems where someone or something has to travel between two or more nodes over arcs between those nodes. This kind of problem applies to many problems. The main idea is that there is a network in an environment. In this network, some nodes and/or arcs have to be visited. Often these problems have many constraints depending on the problem. In several routing problems, like street sweeping, snow-plowing, or salt gritting, specific arcs must be covered, contrary to node routing problems where it is only necessary that all or some nodes are covered (Assad & Golden, 1995).

Some other typical constraints are:

- Time limits to visit a certain number of nodes/arcs.
- The capacity limit for product vehicles can deliver/pick up during the route.
- Money constraint to visit certain nodes/arcs.
- Directed and undirected arcs.
- Opening hours/time budgets of arcs/nodes.
- Number of vehicles/employees

A standard arc routing network consists of a set of R, representing the arcs in network $G = (V, A \cup E)$ with nodes V, directed arcs A, and undirected edges E. It is possible to have a graph G consisting entirely of directed arcs ($E = \emptyset$), of undirected arcs ($A = \emptyset$), or a mixture of both. These three are also known as the Undirected Postman Problem (UPP), the Directed Postman Problem (DPP), and the Mixed Postman Problem (MPP). Some other routing problems are more specific to some instances, these are (Essink, 2014):

- The Rural Postman Problem (RPP) for problems where only a subset of arcs needs to be served instead of all. For example, for delivering posts in rural areas where streets do not have many houses, not every arc has to be visited every day if those houses do not have a post on a day.
- The Stacker Crane Problem (SCP) for problems with the capacity constrain of max one unit. For example, stacker cranes loading containers onto ships or forklifts that only can carry one pallet.
- The Windy Postman Problem (WPP) for problems where the traveling costs or time of the arcs depend on crossing direction. For example, if aircrafts are flying against the direction of the wind or in the same direction. This difference can result in a big difference in time and money saved on fuel.

- The Capacitated Arc Routing Problem (CARP) for similar problems as standard arc routing, but with capacity constraints. This additional constraint makes the problems much more complex to solve.
- The Capacitated Postman Problem (CAPP) for similar problems as RPP's, but with capacity constraints. A prevalent example of this is parcel delivery vans. They can only bring a certain number of packages with them on a route.

2.3.2. Capacitated arc routing problem

The capacitated arc routing problems are similar to the standard arc routing problems but with capacity constraints. In these problems, products have to be picked up or have to be delivered. These products have a volume and weight, and the vehicles have a certain capacity. This constraint makes it more challenging to solve the problem.

Sipahioglu et al. (2010) researched an area covering a problem similar to what we are looking at in this project, where robots had to detect obstacles with sensors. In this so-called multi-robot sensor-based coverage path planning problem (MRSBCP), the Ulusoy partitioning heuristic was modified and used to solve this problem. This heuristic solves fleet size and mixed problems in CARPs for directed and undirected networks (Ulusoy, 1985). MRSBCP is used instead of the traditional CARP because energy consumption is not considered in these cases. Sipahioglu et al. (2010) tested this MRSBCP using a MAS in an indoor room converted into a labyrinth for this study.

2.3.3. Heuristics

Because Lely is not looking for the optimal solution but getting only close to an optimal solution, we will use heuristics instead of looking for the optimal solution. Typically, heuristics are used to solve these arc routing problems. Heuristics have the purpose of giving fast a good solution but not necessarily the optimal solution. Winston (2004) gives two examples of heuristics for arc routing problems, the nearest-neighbor (NNH) and the cheapest-intersection heuristic (CIH).

The NNH begins at any node and then goes to the nearest node that has to be visited. Then the heuristic picks an unvisited node closest to the city that is visited most recently. The heuristic continues in this way until all nodes are visited that have to be visited. The NNH gives not an optimal result. A slight expansion of this heuristic that improves the result is to do the heuristic multiple times, starting with a different node every time.

The CIH starts the same way the NNH does. It also picks some nodes and then goes to the nearest node that has to be visited. The next step is creating a sub tour joining those two nodes following by replacing an arc in the sub tour. For example, arc[1-3], with the combination of two arcs, for example, arc[1-2] and arc[2-3]. We do this with all possible options and put the new value of the KPI in a table. The next step is to pick the best option which harms the KPI the least. This option increases the KPI of the tour by the smallest amount. These steps repeat until all nodes that have to be in the tour are in the tour.

2.3.4. Simultaneous Localization and Mapping

In the previous sub-sections, we discussed the possible heuristics for making routes. For driving a route, the Lely Discovery also uses a heuristic. This heuristic is called Simultaneous Localization And Mapping (SLAM). This heuristic is a heuristic that robots and autonomous vehicles use for routing in mostly enclosed environments. SLAM works with sensors that can recognize points in the environment. With the driven distance, it is possible to say with a particular uncertainty what the location is of a robot or autonomous vehicle (Busoniu & Tamás, 2016).

2.4. Robotics in the agriculture sector: state of the art

Like in every sector, there is a trend in more automated systems and robotics in the agriculture sector, partly because the number of farms decreased and the size of farms increases. For example, the average herd size of dairies in the Netherlands increased from less than 50 in 1996 to around 80 in 2014 (Barkema et al., 2015) and is still growing. This section discusses the state of the art of these systems and robotics—the first part focus on barn solutions, and the second part on field solutions.

2.4.1. Barn solutions

The first automated systems for dairy keeping in barns were developed back in the eighties and early nineties. For example, Lely came with the first fully automated milking system (AMS) in 1995 on the market. The AMS eliminated a big part of the labor a farmer had to conduct every day. With the AMS, the cows' productivity and comfort increased (Wagner-Storch & Palmer, 2003). These kinds of innovations also decreased the physical work a farmer has to do.

In addition to the AMS, there is much development in automatic feeding of cows. These are systems where food for the cows is taken from (temporary) storages, mixed, and served at the feeding spots in the barn. The most significant advantage of this system is that cows can be fed multiple times per day, making the food fresher. This way of feeding has a positive influence on milk production per cow and milk quality.

In many barns, it is common for dairy to walk on concrete floors, slatted or closed. With the slatted floors, most of the manure fall through the slats. The remaining manure is kicked in by the cows or pushed in by a manure scraper. With closed floors, a manure scraper is needed to clean the floor. The scraper can be a path width scraper pulled by a chain, a manual-driven manure sucking machine, or a cleaning robot. The advantage of a scraper pulled by a chain and the cleaning robot is that they can work unmanned. The disadvantage of a scraper pulled by a chain is that it only works in a straight path with a manure pit at the end. Driven manure-sucking machines and cleaning robots do not have to work in straight lines.

For cow health, it is vital to have clean floors. The chance of slipping reduces when the floor is clean, and therefore also the chance of injuries. Besides that, the cows are cleaner, which reduces the chance of infections and bacteria in the milk (Sanaa et al., 1993).

The latest floor and manure management developments are where the urine and feces are split on the floor. The urine flows through a small drain away from the floor. A cleaning robot or scraper cleans the feces. The significant advantage of this splitting is the reduction of ammonia (NH_3). The ammonia comes free from the feces when it gets in contact with the urine (Vaddella et al., 2010).

2.4.2. Field solutions

With the introduction of computer systems in tractors and electronics in the machines, the performance of the machines increased. Systems like RTK-GPS, heatmaps, and yield measurements with NIR sensors are currently the latest developments in agriculture machines. Next to that, there are lots of developments in (semi) autonomous robotics.

Global Positioning System (GPS) is a commonly used system for navigation, also in the agriculture sector. In agriculture, it is used to drive the machine in the fields in straight lines and use the same lines for other operations. The positioning system primarily used in the agriculture sector is RTK-GPS (Real-Time Kinetic GPS). This system has an accuracy of 2cm (Karsten & Tastowe, 2020), where the regular GPS has an accuracy of 30cm. This accuracy makes it possible to drive, plant, harvest and fertilize with high accuracy without damaging crops.

In the last couple of years, there are many developments around Near Infra-Red (NIR) sensors. NIR-sensors can measure the substances and quality of crops, manure, and ground. For example, when harvesters are equipped with NIR sensors, the farmers can directly know the quality of the harvested crops. When combining this technique with the RTK-GPS location, it is possible to create heatmaps with the quality of the yielded crops. This information is beneficial for farmers for the next planting season. With this information, they can adapt their fertilization plan and implement place-specific fertilization to create a more homogeneous yield quality over the field and reduce fertilization.

To go further on the RTK-GPS solutions, if the path of a planter or a harvester is precisely known, it is possible to compute the next row and guide the machine automatically to the correct path belonging to that row after finishing a row. The sequence of actions the machine has to do to get in the next row is pre-programmed. On which level of automation this is, is different for each manufacturer. The significant advantage of these systems is that the turns are done faster, more precise and that the machines are correctly lined up with the rows. This system is called headland management.

2.5. Conclusion

This chapter shows that multi-agent systems are intelligent pieces of software that can be implemented in several environments to solve diverse problems. To design such a MAS, there are multiple methodologies. We have gained insight into a couple of them and decided that the Prometheus methodology is best suited for this project.

To cover the routing problem, we have gained insight into various routing problems. We conclude that arc routing problems are very similar to what we are looking at in this research. After examing various arc routing problems, we conclude that the CARP is the most similar problem compared with ours because we have to consider two different capacities, the battery level of the robot and the remaining capacity for goods. There are multiple heuristics developed to solve these problems fast. The best basis for solving our problem is the NHH with CARP constraints, considering the battery and tank capacity.

3. Prometheus methodology

This chapter contains the elaboration of the Prometheus methodology (Padgham & Winikoff, 2004) as discussed before and answers the following subquestion: *How would a generic self-organizing system look like for agrobotics with simultaneous driving with pickup and delivery?*

The first part of this chapter is on the system specifications. In the second part of this chapter, we discuss the architectural design phase, and in the last part, we discuss the detailed design phase.

3.1. System specifications phase

This section is about the system specifications phase. We go through this phase in systematic order. We first draw up assumptions for the system and how we design it for the Lely case.

Together with Lely, we have set a couple of assumptions and constraints to ensure that the system does not get too big and complex. These assumptions are made for the system's practical functions, which are a must-have in the system. There are also functions defined that are nice to have. The must-haves and nice to have are based on the experience of Lely and the demand of the market. The customers do not want or/and do not need very complex and expensive systems. The eleven assumptions and constraints for the must-haves are summed down here:

1. All available routes are predefined and made by a service technician.
2. All routes are defined so that the manure that has to be picked up during the route will probably not exceed the robot's capacity under typical conditions in the environment. If the capacity is exceeded, the robot continues the route without cleaning up the manure.
3. It is not known how much manure it captures during a route. The robot can only get a message that the tank is full.
4. It is not needed to estimate the expected quantity of manure.
5. All routes start and finish at the charger.
6. The robot dumps the collected manure every time before it goes to the charger.
7. The robot does not need to charge every time it dumps manure.
8. All programmed routes are split into two categories, a set of standard routes and a set of routes used when tasks are shared or taken over.
9. The second set of routes only covers areas that are assumed to have a higher distribution of manure.
10. Robots are only allowed to switch routes when their current route is finished.
11. The system needs to deal with a malfunction in the communication with robots (e.g., a robot is located in a part of the barn with bad connectivity), but we can assume that failing communication is exceptional and not typical behavior.

This list of assumptions and constraints is set up to avoid that the system gets too complex. Later on, we discuss the list of assumptions and constraints we use for the simulation model in this research.

3.1.1. System goal

The first step is to capture what the system should do accurately, that is, what are its goals? To capture the generic goal of the DurableCASE project, we define the following goal:

A **robust automated planning and control system** based on agent technology for **Simultaneous Driving with Pickup and Delivery** (SD-PD) by means of **cooperative vehicles** in a **self-organizing environment** in such a way that it yields a **cost-effective solution**.

To exemplify, we illustrate the system goal and related sub-goals in Table 3-1.

Table 3-1: Overall system goal

Overall system goal	**Lely case**
	A self-organizing barn-floor cleaning system efficiently utilizing cooperative, mobile robots in such a way that it yields a close to the optimal solution
Goal elements	
Robust automated planning and control system	Autonomous manure collecting system with shared routing and cooperative task handovers
Cooperative vehicles	Discovery Collectors
Simultaneous Driving with **P**ickup and **D**elivery	**Discovery** Manure collecting (P) Manure dumping (P/D) Spraying of water (D) Refilling water (P)
Self-organizing environment	Cow barn (indoors)
Cost-effective solution	Timely cleanliness of barn for cow health

Contrary to typical mobile robots, which mainly focus on transporting and or transferring goods from location A to location B. In our focus, the mobile robots perform tasks (e.g., cleaning or collecting) while driving on a place-specific route. We refer to this notion as Simultaneous Driving with Pickup and Delivery (SD-PD). For example, the Discovery can pick up manure and deliver (spraying) water on the cleaned floor.

The overall system goal can be broken down into several goals, as shown in Table 3-2 below.

Table 3-2: System goals

System goals	**Lely case**
Obtain goods	Clean the floor
Deliver goods	Empty robot
Determine route	Determine route
Charge/refuel vehicles	Charge robot
Avoid conflicts	Avoid collisions
	Avoid congestion
Tolerate perturbations	Handle redundancy
	Overcome failure
Schedule vehicles	Schedule resources
Coordinate vehicles	Handover task
Replenish vehicles	Fill water tank

3.1.2. Scenarios

Robots with system goals as described in Section 3.1.1 are acting similarly in different applications. Because of that, we can make different scenarios (Table 3-3).

Table 3-3: System scenarios

System scenarios	Lely case
Start vehicle	Start vehicle
Stop vehicle	Stop vehicle
Start task	Start cleaning
Stop task	Stop cleaning
Capacity reached	Vehicle full
Empty	Vehicle emptying
Battery/fuel low	Vehicle battery low
Charge/refuel	Vehicle charging
Schedule	Schedule task
Coordinate	Reallocate task
Failure	Vehicle failed
Reboot	Vehicle rebooted
Conflict	Vehicle in conflict
Refill	Fill water tank

3.1.3. Percepts

Percepts are messages that contain information about the status of the robot. The agents send these percepts so other agents can respond to them. The percepts are mentioned in Table 3-4. Later in this chapter, we describe each percept in detail.

Table 3-4: Percepts

Lely case	
Manure tank full	Water tank full
Manure tank empty	Route finished
Battery level low	Connection lost
Battery full	Delivery point occupied
Vehicle stopped	Charging station occupied
Vehicle stuck	Location unknown
Water tank empty	

3.1.4. Actions

Actions are the responses of the agents to a percept. It can be seen as if-statements; if something happens, then something has to be done. The actions that are defined for this project can be found in Table 3-5.

Table 3-5: Actions

Lely case	
Request delivery spot	Request route
Stop driving	Open tank
Start driving	Close tank
Reallocate task	

3.1.5. Functionalities

A functionality is a piece of behavior that a robot can have. This part describes all the functionalities with their goals, actions, and triggers. In this project, 14 different functionalities are defined (Table 3-6). One of the functionalities is worked out in this report (Table 3-7). In this example, all the functionality attributes are the same for both cases, except one. That exception is due to machine specifications and different software and hardware systems and is the case for multiple functionalities. All of the other functionalities can be found in Appendix A.

Table 3-6: Functionalities

Functionalities	
Determining remaining capacity	Determining fuel/battery level
Obstacle detection	Determining location
Pause/resume	Determine next route
Retrieve/collect goods	Dump goods
Go to start/end location (charging point)	Increase/decrease of active robots in the system
Route adaption	Cooperative dispatching
Forecasting	Monitoring and logging

Table 3-7: Determining location functionality

Determining location	**Lely case**
Description	This functionality localizes the robot in the environment where it is currently working
Goal	To get to know the location of the robot
Actions	Continue working or change job-based on the location
Triggers	Continuous measurement
Information used	Information from multiple sensors and usage of the SLAM algorithm
Information produced	The location of the robot

3.2. Architectural Design phase

As the name suggests, this section contains the architectural design phase. This phase determines which agents are used in the system and in which environment the agents operate. Also, several diagrams capture the system's overall structure and describe the system's dynamic behavior.

3.2.1. Agent types

To determine which types of agents are used, we are going to group the functionalities. This grouping of functionalities is done based on clear logic and shared data. There has to be a good balance between the number of agents and the complexity of an agent to create a robust system that can still perform its main task without a proper functioning agent or communication with a central system or another agent. The number of agents that are designed must not be too small or too large. If we take the extreme with picking for every functionality of an agent, it results in a very high dependency between the agents. On the other extreme, where we put all the functionalities under one agent, agents lack cohesion because unrelated functionalities are grouped and become too complex.

All functionalities are gathered in Figure 3-1 together with the data they need and produce. There are three different kinds of functionalities and data in the figure and are marked in different colors. This categorizing is based on the wishes of Lely. The white functionalities and data are already present in the current system, and the green ones are implemented to ensure that they cover the leading root causes. The last ones are the oranges ones; these are needed to cover the system functions, which are nice to have.

All data and functionalities are divided into three levels: high-level, mid-level, and low-level. Everything is arranged in one of the levels depending on the frequency of updating the data or the frequency of using the agent. The high level is mainly external (cloud) data. In the mid-level are the more complicated agents living. The data used and produced at this level is not continuously communicated. At the low level are the agents living with the basic functionalities needed to let the robot operate at all times.

The whole system consists of six agents with each their own function. All the properties of each agent can be found in Appendix B. The Vehicle Operating Agent, the Vehicle Status Agent, and the Location Agent live on the low level. Those three agents cover the basic functionalities of the robot. The robots perform their primary tasks with these functionalities.

The Location Agent responds to the information of the location sensors and determines the location based on that information. This location data passes the Location Agent to the Vehicle Operating Agent, the Cooperative Agent, and the Monitoring and Logging Agent. Furthermore, it checks for obstacles on its path.

The Vehicle Operating Agent is responsible for keeping track of the remaining capacity and the robot's remaining battery or fuel level. This information is based on the information from fill level sensors and battery sensors. Based on this, the Vehicle Status Agent can pass on the remaining capacity and remaining working time of the robot to the Vehicle Operating Agent and the Monitoring and Logging Agent.

The Vehicle Operating Agent is the agent that controls the robot. In the typical situation, this agent uses the information from the Location Agent and the Vehicle Status Agent and makes decisions based on the information and user input. Decisions that are made are:

- Pause or resume the current job.
- Determine next route
- Retrieve or collect goods
- Dump goods
- Go to the start/end location

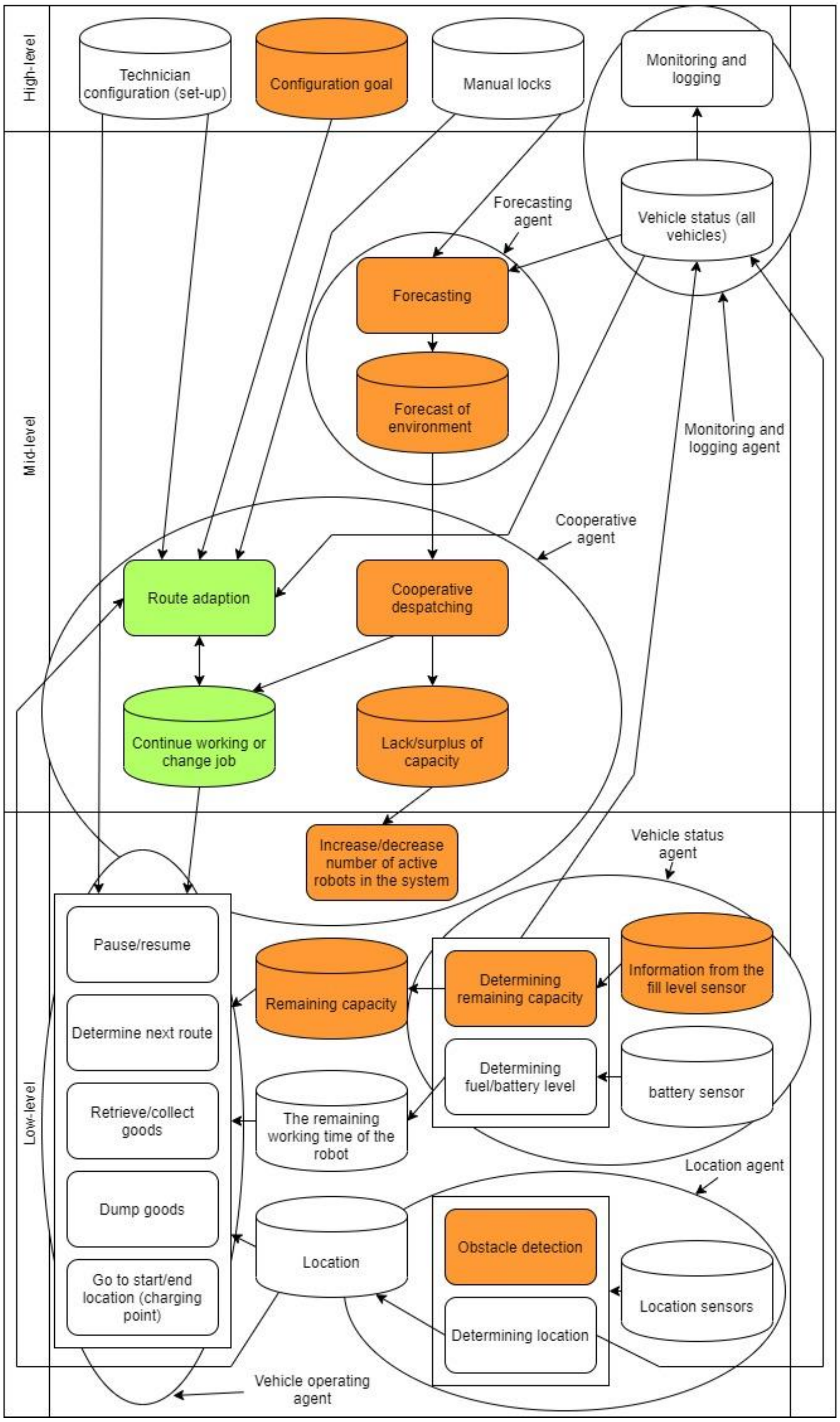
When the situation is different from usual in the environment of the robots, the robot also gets input from the Cooperative Agent.

In the mid-level are two agents living, the Cooperative Agent and the Forecasting Agent. Those two agents can make advanced decisions based on the current state of multiple robots and the environment.

The Forecasting Agent can forecast the environment based on information from the robots collected in the surrounding areas, history, and human input. The quantity of manure in a specific lane is an example of what can be forecasted. The Forecasting Agent passed this forecast to the Cooperative Agent.

When there is a disruption in the system, the Cooperative Agent can calculate a suitable, close to the optimal solution. Disruption can be a quick change of the environment or a failure with a robot. The agent calculates what the best next thing is to do for the robots. The best next thing to do is based on the last known information it can receive from the Monitoring and Logging Agent and the forecast that the Forecasting Agent has made.

At the High level is everything that situates outside the environment (e.g., cloud connection). This level mainly consists of user input. Input examples are the system set up, the configuration goal set by the user, and if parts of the environment are locked. Also, the Monitoring and Logging Agent is living on this level. As the name suggests, this agent's main task is to log and monitor the robots in the environment.



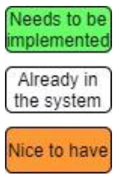


Figure 3-1: Multi-Agent System model

3.2.2. Agent approach

Different communication strategies are needed to keep the designed MAS as generic as possible and suitable for different environments (e.g., barns, open fields with crops). Therefore, we have designed three different approaches to put “the cleverness” of the system. The three different approaches are a central approach, a decentral approach, and a hybrid approach. All approaches have their pros and cons. These are summarized in Appendix C. Per case, it is different which approach is applicable and the best fit for that specific case. It is possible to have the standard static routing and neighborhood checking for conflict avoidance in shared places in all approaches. Down here, there is a more extensive description of all three approaches.

Central approach

The first approach we are going to discuss is the central approach (Figure 3-2). In this approach, intelligent and advanced agents live in a central place, and the robots themselves are only equipped with the agents controlling the basic functionalities. In this approach, the robots can only use the capabilities of the advanced agents if the robots have a connection with the central system. When the robots have a continuous connection, it is possible to have a dynamic routing system with continuous optimization routes for all robots, with only a low number of conflicts and a high number of route adaptions. The big downside of this approach is that much communication is needed between the central system and robots, which also takes on the system's security. Communication can be challenging due to the environment where the robots are active. These environments are often located in remote places with bad communication connections or barns with many metal fences and are dirty, disrupting the signal. Because of that, there is a high chance of a Single Point Of Failure (SPOF) for the advanced functionalities. For the basic functionalities, there is only a low chance of SPOF.

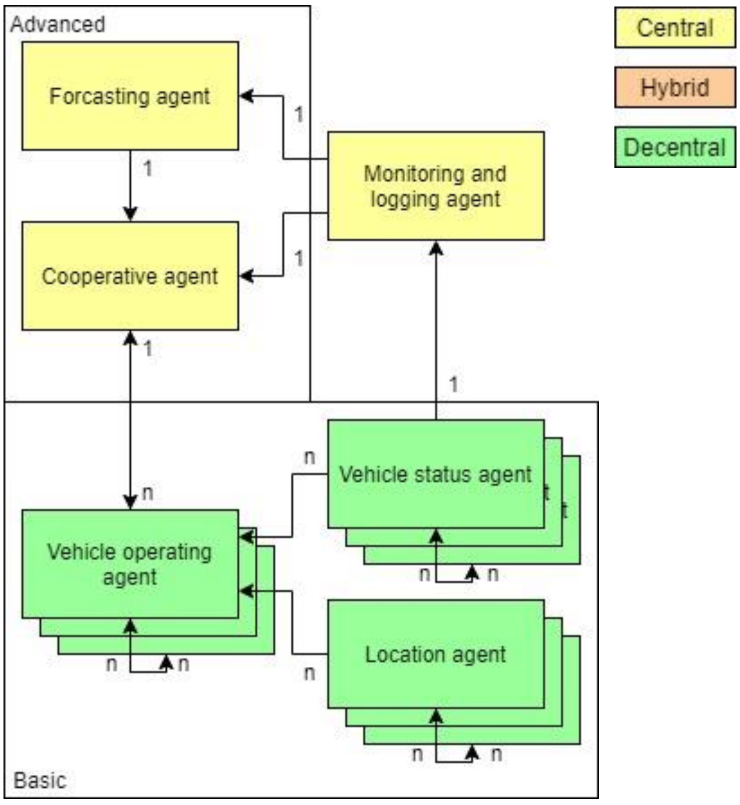


Figure 3-2: Agent coupling diagram central approach

Decentral approach

The second approach we are going to discuss is the decentral approach (Figure 3-3). In this approach, the intelligent and advanced agents live on the robots themselves instead of in a central place. Placing intelligence like this means that all the robots have all the capabilities of the system living on themself and can make advanced decisions by themself. The robots can choose to form a different set of routes when they observe a failure at another robot. That information has to be sent from a robot with a failure. In this approach, it is hard to optimize the whole system because there is no leader in the system. Another downside of this approach is that it cannot adapt to the other robots when it has to decide what to do next and not contact others. Therefore, the number of conflicts is much higher compared with the central approach. Also, the system's performance is lower since there is less communication between the different entities in the system. But this has a possible effect on the system's security because this system needs way fewer messages between the different entities.

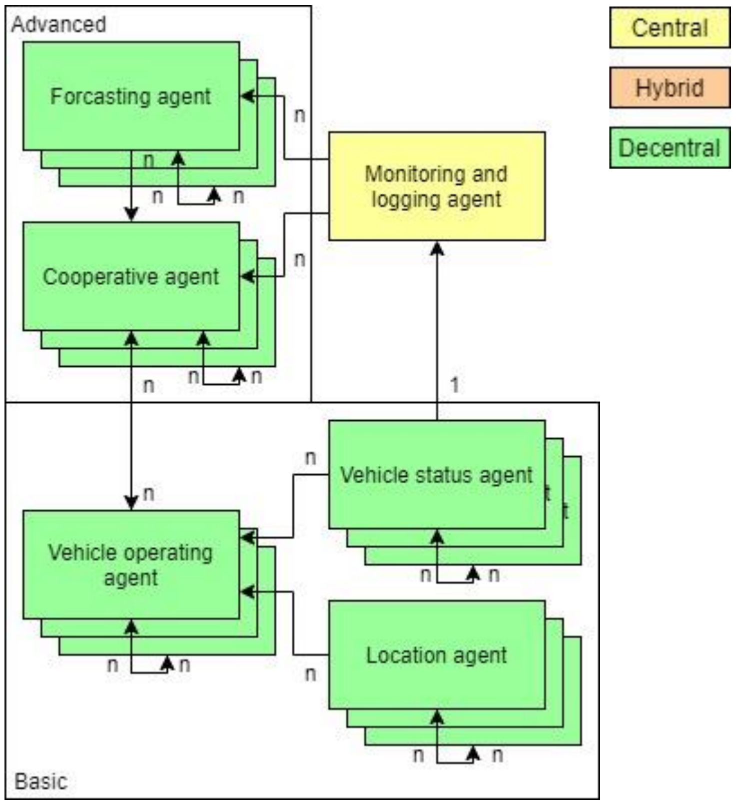


Figure 3-3: Agent coupling diagram decentral approach

Hybrid approach

The third and last approach we discuss is the hybrid approach (Figure 3-4). This approach combines the central and the decentral approach. The intelligence in this approach is divided over the robots and the central system. Like the decentral approach, the robots can communicate with each other when they are nearby. But in this approach, the robots' intelligence is only capable of essential conflict-avoidance and basic route adaption. Next, they can bring information about the other robots to the central system where the remaining part of the intelligence is. The advanced agents process this information and determine which task and routes the other robots have to take. An advantage of this system is that the central system takes all robots into account when assigning new routes to cover the failure in the system.

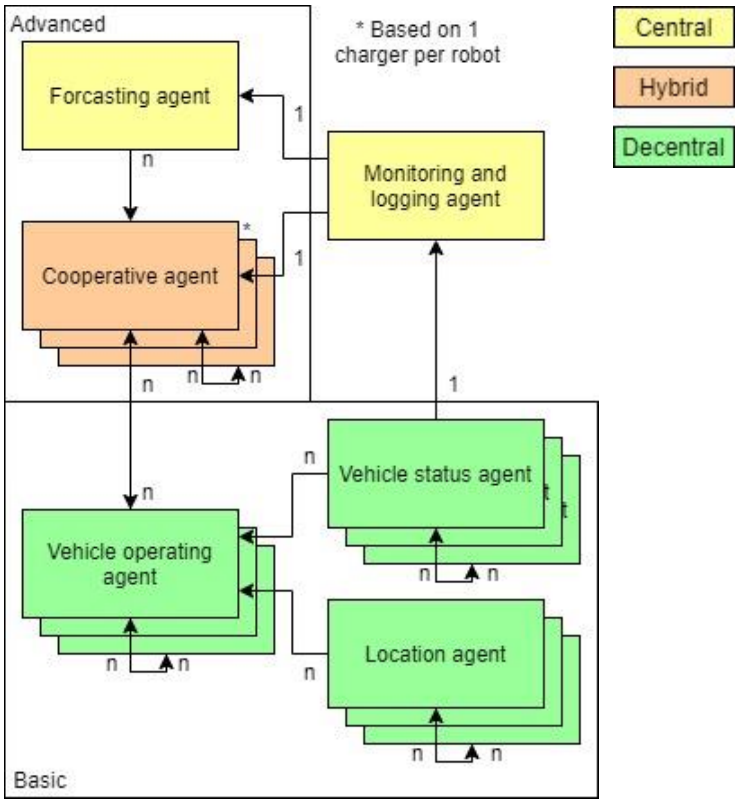


Figure 3-4: Agent coupling diagram hybrid approach

After discussing the different approaches with the consortium partners, we have decided to focus on the hybrid approach in this project. We decided because it is hard to connect the robots and the central system and the robots themselves in the robots' environment. With the hybrid approach, we can make sure that the robots can take care of themself in case it is needed. And in this case, we can create a physical connection between the robot and the central system through the charger.

3.2.3. Agent interactions

To get a good overview of how, when, and in which sequence the interactions between the agents occur, we have made an interaction diagram with the robot's activities. In these diagrams, all interactions between the agents are visualized in the correct order. Based on the outcome of messages passing through, different alternatives are selected. Because we focus on the hybrid approach in this research, the interaction diagram of only this approach is shown below in Figure 3-5. The interaction diagrams of the other approaches can be found in Appendix D.

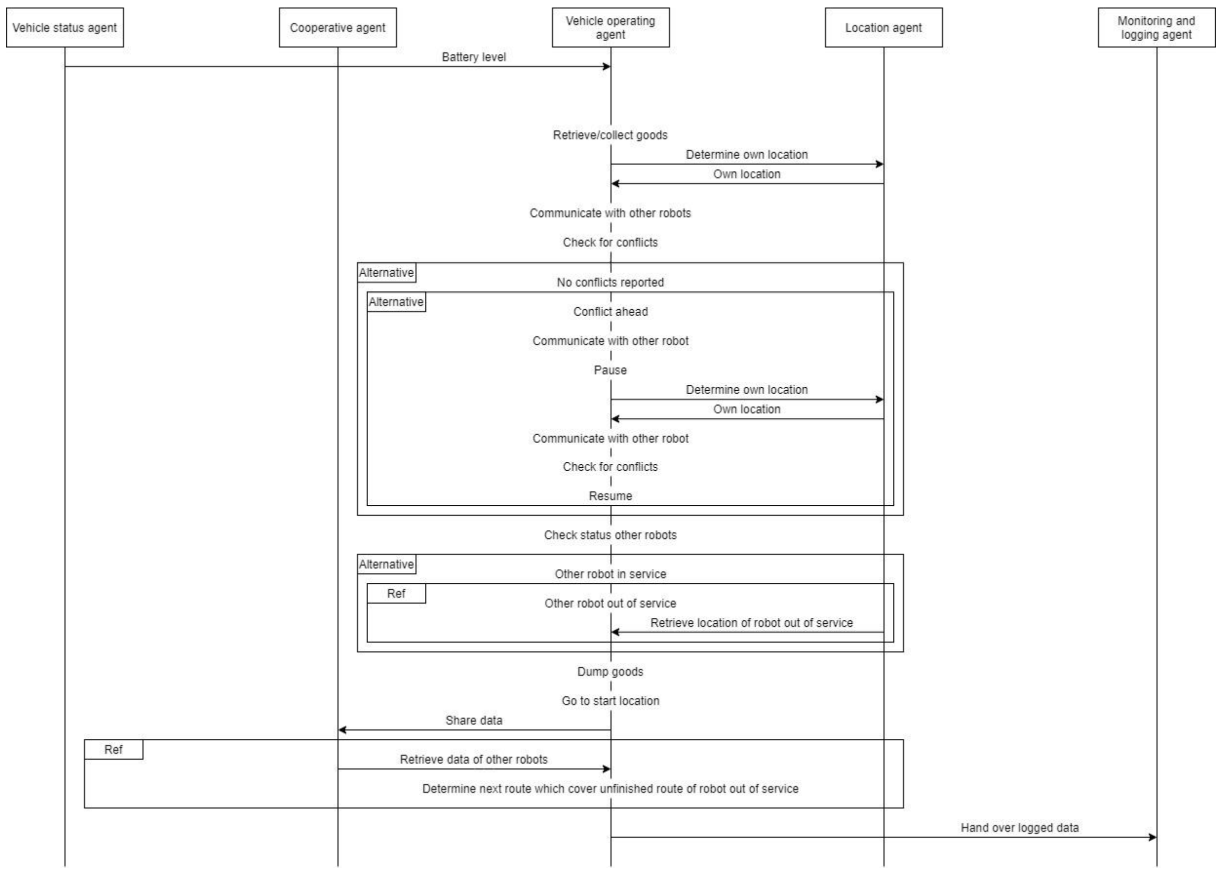


Figure 3-5: Agent interaction diagram hybrid approach

3.2.4. Message descriptors

To pursue the system goal, the agents have to cooperate, and therefore the agents have to communicate with each other by sending messages with information. The message descriptors are made to clarify which messages there are and what these messages have to contain. Table 3-8 is an example of one of the messages, an overview of all descriptors can be found in Appendix E.

Table 3-8: Message descriptor Vehicle Status

Name	**Vehicle status**
Description	The status of the vehicles (battery level, remaining capacity)
From agent	Vehicle status agent, monitoring, and logging agent
To agent	Monitoring and logging agent, forecasting agent, cooperative agent
Purpose	To share the current vehicle status for logging, monitoring, forecasting, and optimize the system
Information carried	Battery level, remaining capacity

3.2.5. Protocols

In protocols are all the interactions described between the agents. Based on these interactions, agents decide what to do. A protocol can contain multiple messages between agents. Table 3-9 is an example of one of the protocols. All the protocols can be found in Appendix F.

Table 3-9: Cooperative dispatching protocol

Name	**Cooperative dispatching protocol**
Description	This protocol gives the command to robots if they have to change their current working routine/job
Scenarios	Start vehicle, stop vehicle, start task, stop task, schedule, coordinate
Agents	Cooperative agent, vehicle operating agent
Messages	Vehicle status, location of the vehicle
Notes	It depends on the type of system (e.g., central, decentral, or hybrid) in which scenarios, agents, and messages are used.

3.2.6. System Overview

In this section, we identify the boundaries of the MAS and the interactions with other sub-systems. Secondly, we describe the percepts, actions, and relationships between these and relevant agents. Thirdly we define all shared data, both persistent external data and internal shared data within the system. In the end, we develop the system overview diagram.

System Boundaries

The system consists of six different types of agents, with each their own tasks. The first three agents we describe here are the agents living on each robot and controlling the robot's primary tasks. These tasks are similar to the current functionalities of the robots how they are working now in barns. These agents can be considered low-level agents. The first one is the Vehicle Status Agent, and this one keeps track of the remaining capacity of the robot and battery level. This data is communicated with the second agent, the Vehicle Operating Agent. This agent controls the main tasks (e.g., cleaning, dumping) of the robot. The third agent is the Location Agent and keeps track of the robot's location and makes sure that the robot does not ride into obstacles.

The last three agents are the agents that are not living on the robot. These agents are placed in the mid-level and high-level. The first one of those three is the Monitoring and logging Agent. As the name suggests, this agent logs and monitors all data of the robots and combines it with the vehicle status of each robot. The second one is the Forecasting Agent. This agent forecasts how the distribution of the manure is in the coming period. The last agent is the Cooperative Agent. This agent is the ‘smart’ agent of the whole system and decides if robots have to be allocated elsewhere or different in the system or keep going with their regular schedule and tasks.

Percepts

In this part, we give a detailed description of the precepts that the robots sent. The information is given in a table for every percept. In this stage, we are only focusing on the precepts of the Lely case. In Table 3-10, the description of the percept “manure tank full” is shown. All the descriptions of all other percepts can be found in Appendix G.

Table 3-10: Percept Manure tank full

Name	**Manure tank full**
Description	It occurs when the manure tank of the robot is full
Information carried	That the manure tank of the robot has reached its capacity and the robot cannot pick up any manure anymore
Knowledge updated	One of the active robots cannot pick up any manure anymore temporarily.
Source	Information comes from the manure level sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	After a specific time of working

Actions

For the actions, we use a similar approach to describe them as the percepts. The actions can be information-based (e.g., Request delivery spot) and motion-based (e.g., open manure tank). Again, these actions are only for the Lely case, and one example is shown (Table 3-11), and all the action descriptions can be found in Appendix H.

Table 3-11: Action Request delivery spot

Name	**Request delivery spot**
Description	With this action, a robot can request a delivery spot to deliver its goods. The result is that the robot gets a spot assigned where it can go to deliver its goods.
Parameters	The location of the robot and the other robot. If the delivery spots are taken or not.
Temporality	Instantaneous
Failure detection	Yes, could not assign a delivery spot
Partial change	The robot has to wait for a while and take another attempt
Side effects	None

We have made a system overview diagram to understand where and how information is shared and processed (Figure 3-6). In this diagram are several icons, each with a different type of part in the system. In this diagram, we have used the color green for the parts that are entirely new in the system of the Lely barn cleaner and are needed for the must-haves, and the color orange for the parts that are nice to have.

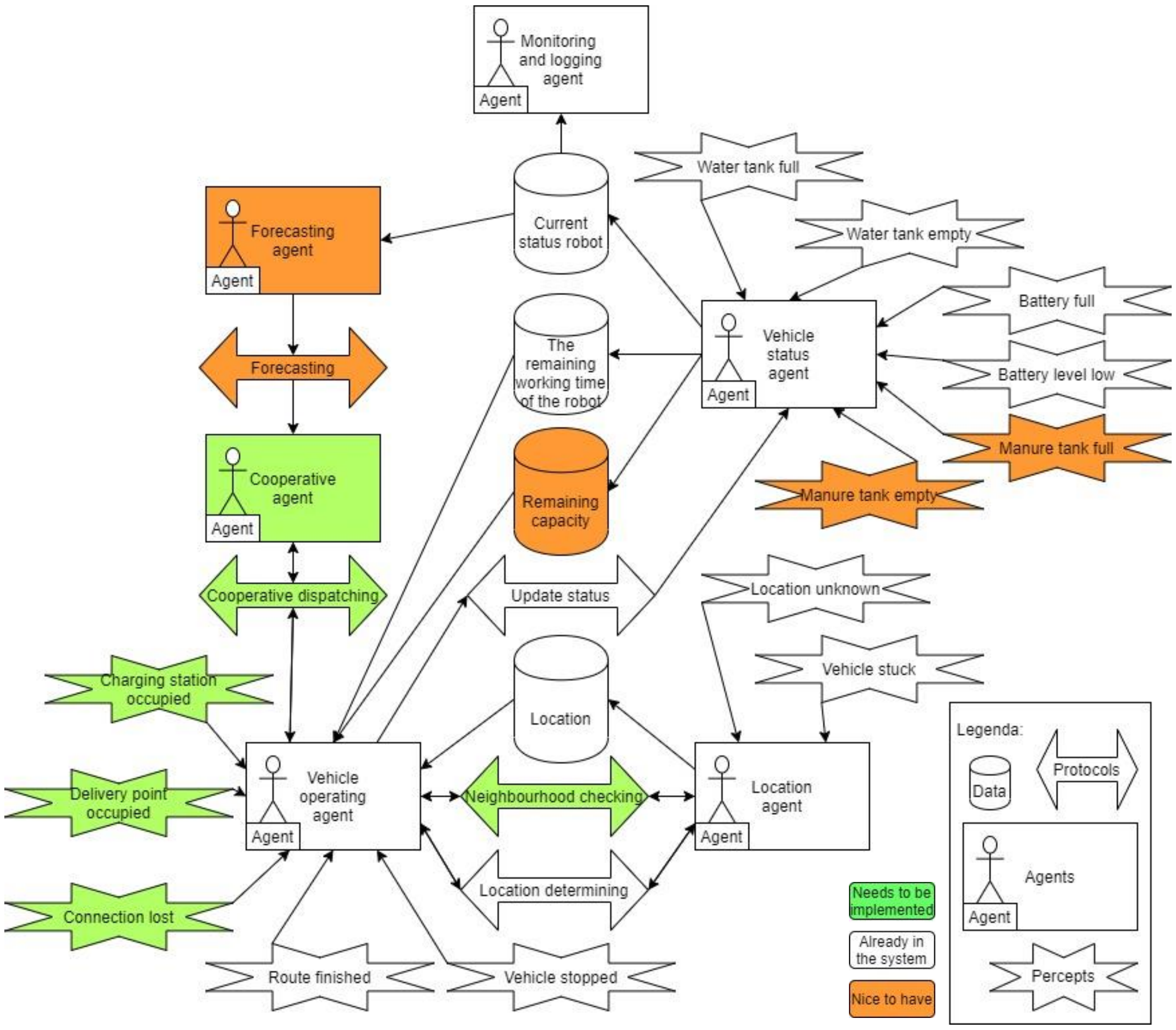


Figure 3-6: System overview diagram

3.3. Detailed Design phase

In this section, we discuss the last details about the MAS design. In the first part, we discuss the capabilities the agents need to fulfill their tasks and summarize these capabilities in capabilities descriptors. We make the relations between them visual with the agent overview diagram and the capability overview diagrams. At last, we visualize how the robots react to particular messages and which actions they undertake with logic flowcharts.

3.3.1. Agent overview

In this subsection, we discuss all parts where the system is capable of. First, the system overview diagram (Figure 3-6) is broken down, focusing on every agent and its capabilities. Table 3-12 to Table 3-17 are all capabilities summarized per agent with their goal. Agent overview diagrams visualize how the data, percepts, and messages are transferred within every agent. Figure 3-7 is the agent overview diagram shown of the Vehicle Operating Agent. All other agent overview diagrams can be found in Appendix I.

Table 3-12: Capabilities Vehicle Operating Agent

Name	**Vehicle operating agent**
Routing managing	Goal: selecting the next route for the robot
Collecting	Goal: collect all the manure on his path
Dumping	Goal: dump the collected manure on the correct point
Charging	Goal: charge the vehicle

Table 3-13: Capabilities Vehicle Status Agent

Name	**Vehicle status agent**
Capacity managing	Goal: monitoring the remaining capacity of the manure tank in the robot and communicate as soon it is almost full
Battery managing	Goal: monitoring the remaining capacity of the battery in the robot and communicate as soon it is almost empty

Table 3-14: Capabilities Location Agent

Name	**Location agent**
Location managing	Goal: monitoring the current location and communicate when it is asked for

Table 3-15: Capabilities Cooperative Agent

Name	**Cooperative agent**
Cooperative dispatching	Goal: dispatch the robot's task in such a way that the system keeps performing without or with mirror losses on the KPIs based on the information given by the forecasting agent
Route adaption	Goal: to adapt the routes in such a way that the system keeps performing without or with mirror losses on the KPIs based on the information given by the vehicle operating agent

Table 3-16: Capabilities Forecasting agent

Name	**Forecasting agent**
Forecasting	Goal: making a forecast of the environments

Table 3-17: Capabilities Monitoring and logging agent

Name	**Monitoring and logging agent**
Monitoring	Goal: monitor the current status of all the robots
Logging	Goal: log all the status and performance of all the robots

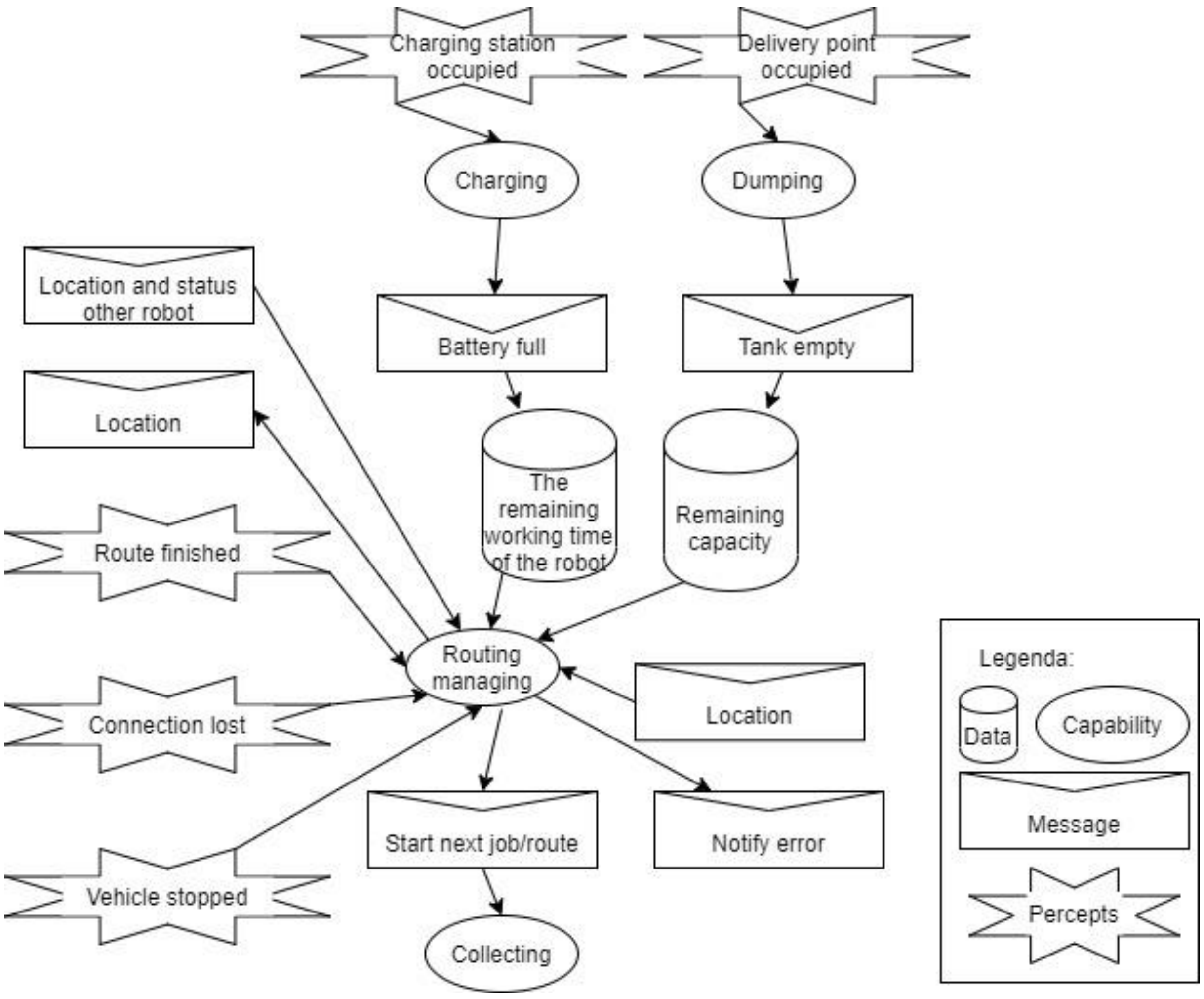


Figure 3-7: Agent overview diagram Vehicle Operating Agent

3.3.2. Capability overview

In this subsection, we dive again one step further in the system by breaking down the capabilities. This step is done by making capabilities descriptors of every capability in the system In Table 3-18 is an example of the capability descriptor routing managing shown. All other descriptors can be found in Appendix J. These descriptions are visualized in capability overview diagrams. Again, an example of routing managing can be found in Figure 3-8, and all other capability overview diagrams can be found in Appendix K.

Table 3-18: Capability descriptor routing managing

Name	**Routing managing**
Description	This capability selects the next route for the robot based on the information it got
Goals	selecting the next route for the robot
Processes	Determining the next route
Protocols	Cooperative dispatching
Outgoing messages	Start route x
Incoming messages	Battery full, Tank empty
Internal messages	Which sectors are going to be combined to create the next route, which sectors are cleaned in total, and which is a pass-through sector
Percepts	Route finished, connection lost, Vehicle stopped
Actions	Select the next route
Included capabilities	-
Data used: imported	Battery level, status other robots
Data internal	-
Included capabilities	Checkup other robots, Error handling, Schedule check
Included plans	-
Notes	-

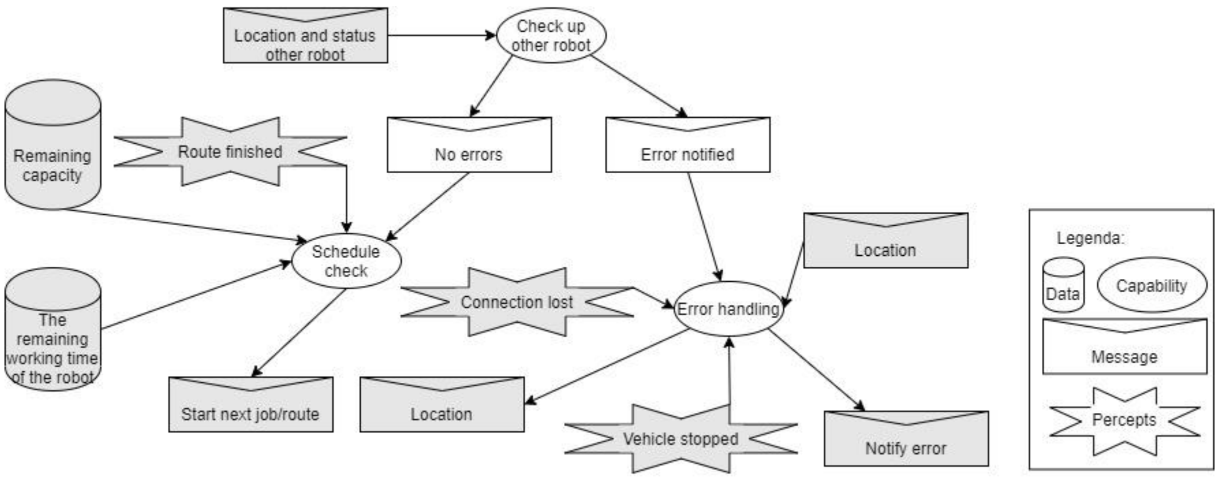


Figure 3-8: Capability diagram routing managing

3.4. Decisions capabilities

In this section, we focus on the decisions capabilities of the agents. For most decisions, there are multiple strategies to make decisions. We discuss these decision-making strategies per agent.

The Monitoring and Logging Agent

The primary purpose of the Monitoring and Logging Agent is to keep track of the status of the environment. The decisions this agent has to make are blocking sectors and facilities. The decisions this agent makes are based on information from the low-level agents.

The Forecasting Agent

The Forecasting Agent is responsible for making a forecast of the environment. Forecasting can be done in multiple ways. The easiest way of forecasting is based on the last visiting time, assuming that the increase of manure per area is the same over time. So the area with the longest time since the last visit is the most urgent to clean. This strategy is used in our model.

A more advanced strategy is to link the manure distribution to several function areas and give weights on time since the last visit. In this way, function areas with higher distribution get cleaned faster. An extension of this strategy is implementing self-learning robots. Self-learning robots make it is possible to update the weights to optimal for better system performance. The updating is possible by keeping track of when and how much manure is picked up from the floor.

The Cooperative Agent

The Cooperative Agent is the most intelligent agent of all the agents in the system and has the most decision capabilities. For the decisions is much information is needed from different agents. Based on the current status of the environment, this agent decides which robot has to clean the most urgent area. The information about the most urgent area to clean is received from the Forecasting Agent, and which robots are available from the Monitoring and Logging Agent. With this information, the agent can provide the new route the robot has to drive.

The Vehicle Operating Agent

The Vehicle Operating Agent is responsible for the basic functionalities of the robot. For this agent are the most of the tasks pre-programmed and scheduled. This agent's decision is where to go if the route needs to be adapted to avoid a collision during a route.

The Robot Status Agent

The Robot Status Agent keeps track of all the capacities of the robot. Several decisions are made on this information. Our model handles two different types of capacities: the battery level and the tank's fill level.

Several strategies are possible for battery management. The first one is to always charge the battery to the maximum or a pre-set level. The second one is to wait till the battery is charged enough to complete its next job. This strategy is used in our model for the take-over routes. A safety factor is used in the model to ensure that the robots make it back to the charger. The last one is similar to the second one. In this strategy, there has also an event be planned to do for the robot. So when there is a trigger to start a new route, there is a check if the battery is charged enough to finish the route. This strategy is used in our model to start a route from the regular schedule with the addition that it can start later within a specific time window if the battery is not charged enough.

For the capacity are two strategies possible. The first one is just simple to keep going until the maximum capacity is reached. The second one has two variants. A simple one where the goods are dumped at the end of a route or lane. This strategy is used in our model, except the dumps are during a route. A more extensive version of this strategy is to combine it with dynamic route planning of multiple robots. It could be beneficial to dump goods earlier because of traveling time to a dump spot. The dumping of goods could be combined with early handover to other robots if this positively affects the performance.

The Location Agent

The Location Agent's only decision is if the robot needs to adapt the route or not. A robot needs to adapt its route if a collision with another robot is near. This decision is made together with the Location Agent of the other robot.

3.5. Conclusion

In this chapter, we have designed a generic MAS, which is applicable for different cases where communication is not always possible, and the main task of the robots is delivering and/or picking up while driving at a specific place and time. The system consists of six agents, which are placed over a high, mid, and low level:

- The Monitoring and Logging Agent (high-level)
- The Forecasting Agent (high-level)
- The Cooperative Agent (mid-level)
- The Vehicle Operating Agent (low-level)
- The Robot Status Agent (low-level)
- The Location Agent (low-level)

Based on the specific case and the communication possibilities, is it possible to place the intelligence in different places. The MAS is designed to be implemented as a central approach, a decentral approach, or a hybrid approach.

4. Use case description

This chapter discusses how we test the usefulness of the MAS that we have designed in the previous chapter. In Chapter 1, we already introduced the use case and the problems that are currently occurring. This chapter gives a more detailed and technical description of the barn used in the use case, the Lely Discovery 120 collector, and the cows. In the last part of this chapter, we discuss the scenarios in full detail with the heuristics. Together, we can partly answer the sub-questing: *How would a simulation model look like for a self-organizing system for barn cleaning robots?*

4.1. Use case location

The barn which we are going to use in our use case is one of the testing barns of Lely. This barn is equipped with two Lely Discovery barn cleaning robots and two Lely Astronaut A4 AMS. The barn can hold 120 cows, based on the joined capacity of the two AMS (each AMS has a capacity of 60 cows). The barn has a walking surface of $591.46m^2$ that needs to be cleaned, 66 meters of fence where cows can eat divided over two equal pieces, two concentrate feeders, and 94 cubicles where cows can lay down. The barn has a closed floor. A blueprint of the barn can be found in Appendix L.

4.2. Lely Discovery 120

The Lely Discovery 120 is an autonomous barn cleaning robot for cleaning the manure from barn floors. The robot follows a fixed schedule when to start which route. A service technician of Lely preprograms the routes which the robots are driving. While driving, the robot is continuously picking up manure and can additionally spray water on the floor. The water spraying gives a cleaner result and makes sure that manure cannot dry and gets stuck to the floor.

The robot drives at a speed of $0.15 \, {}^{m}/_{sec}$ through the barn and has a working width of $120cm$. The maximum capacity of the manure tank is $320l$ when not carrying water. When carrying water, the capacity is less because the water is stored in bags inside the manure tank. The power usage and charging capacity are equal to each other, and this means that one minute of charging gives the robot the power of one minute of driving and cleaning. The maximum capacity of the robot's battery gives the power to clean and drive for 50 minutes straight.

The robot determines its location with the SMART heuristic. And it moves through the barn based on a sequence of commands. Commands can be, for example: rotate 90 degrees or drive forward until 100 wheel rotations are reached. The downside of this heuristic is that the robot cannot know precisely where it is, but only with a certain accuracy. This inaccuracy is due to wheelspin and bumping into obstacles. The accuracy gets lower as the commands are getting longer. The robot can bump slowly into recognizing points in the barn during these commands, like walls and fences. When this happens, the robot knows where it is precisely, and the accuracy of the location is high again.

It is possible to have a combined dumping spot with a charger and refilling station for water for the robots. It is also possible for the robot to dump their manure when driving over slatted floors. In the barn used in this study, the dumping spots and charging points are at different locations in the barn. Both of the robots which are working in the barn have their dump spot and charging station.

4.3. Cows

The cows that are living in the barn are producing the manure that the robots have to clean. The amount of manure a cow produces on a day depends on different factors, like the cow's age, the cow's diet, and the time passed since the last time of giving birth to a calf. On average, a cow produces $80l$ per day. Also, a significant impact on the barn’s cleanliness is if the cows can go outside to the meadow. The herd distribution is uniform in the case study, and the cows cannot go outside the barn. Also, the distribution of manure production is uniform over time and place for simplicity. To make the manure production per location in the barn more realistic, we created three different levels for the various functional areas in the barn. The first level of areas has a slightly higher increase of manure (e.g., close to the cubicles) than the second level (e.g., the pathways) and third level (e.g., close to the feeding fence). Which area is given which level is based on logic thinking, based on own experience, the experience of experts, on the research of Villettaz Robichaud et al., (2011), and a route analysis of the current routes are driven by the robots. This route analysis is discussed in detail in Section 6.2.

4.4. Scenarios

With the simulation study, we want to test the performance and usability of the MAS. During the simulation study, we test three scenarios at the same time. The first scenario is implementing shared places and facilities. In the current situation, every robot has its own charging station and its own manure dumping spot. Also, the routes are currently programmed so that they never cross each other’s paths. In this scenario, it is possible that robots can come along each other’s path and use their dumping spot. The charging location is still owned by one of the robots for practical reasons.

The second scenario is the scenario where task handovers are implemented. In this scenario, a task handover is that the robots temporarily do not drive their routes according to the schedule but drive routes that cover the areas at that moment in time the most important to do based on the current state of the environment. This situation happens when a robot gets a failure and communicated this to the central system. This communication can be done in two different ways, firstly it can communicate within a specific range of particular points (e.g., at the charging stations, nearby the AMS) in the barn, connected with the internet. Secondly, it can communicate with other robots which are in a specific range. Another robot can bring the failure message to the central system. The information shared between the robot is fundamental and consists only of the location, the route the robot was driving, and the time of the message. The central system is also capable of sending information back, such as new route schedules. A heuristic makes these additional routes based on the NNH (Winston, 2004). For this heuristic, the barn is split up into different sectors (Figure 4-1). Every sector can be seen as an arc where can be traveled on. At both ends of the arc, we have a node. There are two different routes for every arc, which can be traveled from both sides, so the arcs are undirected. The first type of route is a cleaning route where the total surface of the arc is cleaned. The second type of route is a crossing route. This type of route is taking the shortest path between the two nodes without cleaning the whole arc.

When implementing the second scenario, the first scenario is also needed. Because in the second scenario, the robots are driving in the area where they usually are not driving.

The third scenario is the fully autonomous scenario without pre-programmed routes. This means that each route the AGVs drive is explicitly determined on the status of the environment at that point in time. So, the sector with the highest urgency is always cleaned first. In this scenario, the arcs connected to the node of the charging point can only be visited by the AGV, which is assigned to that charging point. Also, to reduce unnecessary traveling, the arcs, which are entirely on the opposite side of the barn, can not be chosen to clean by an AGV. These two rules are not holding when one of the AGV has a failure.

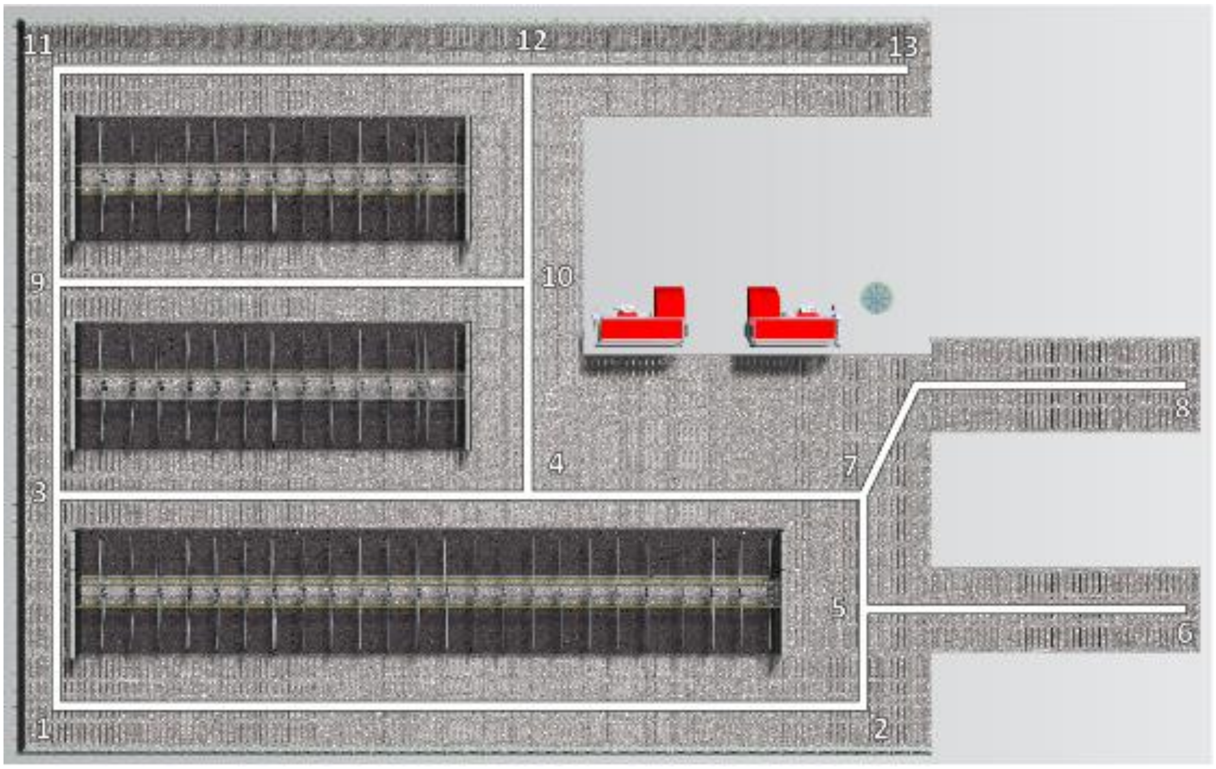


Figure 4-1: Visualization of the arcs and nodes in the barn

In all scenarios, a robot may be driving his route and comes across a robot with an error blocking his path. In this situation, the robot driving adapts its route locally by driving from point to point on a virtual grid. Which point it drives to is based on the distances between all local points and the end destination. A no-go list of blocked points or worse solutions is kept to prevent the robot from getting stuck in a loop.

4.5. Heuristics

This section introduces the two heuristics that are used in the model to make and adapt the routes. The first one is for making entirely new routes. These routes are close to optimal to do in that point of time based on the current status of the environment. The second one is collision avoidance when a robot comes across a robot with unknown failure.

4.5.1. Optimization heuristic

Based on the NNH with CARP constraints, the optimization heuristic (Figure 4-2) is responsible for creating entirely new routes when a new route is needed. As input for this heuristic is the most urgent sector to clean and the AGV assigned to the job needed. Other agents are determining this information. Section 5.6 explains how this is done. The capacity constraints in this heuristic are working differently than in the standard CARP. Instead of making a route fitting the current battery level, we calculate the needed battery level for the route that is made and set a constraint for a starting time when the AGV has charged enough to complete the route. The maximum battery level is high enough to cover all possible routes with the constraint to cover only one sector to clean.

The first part is selecting the starting node. This node is the closest node to the charger of the AGV going to do the job. The next step is to loop over a procedure while one of the sector nodes to clean is not reached jet. So, the procedure keeps on adding nodes to the route till the sector to clean is reached. In this procedure, each step looks at the possible next best node to which the AGV can travel. These nodes have to be adjacent to the current node. Which node is the best node is based on the shortest Euclidean distance between that node and one of the nodes of the sector to clean.

When a new node is selected, there is a check if the route is not going into a dead-end or is getting stuck in a loop. If this is not the case, there is a check if the sector to clean is already reached. If not, the procedure starts again. If so, the route goes on to a dump spot or back to the charging station. The route only goes directly back to the charger if the sector to clean is connected with a dump spot. Going to the dump spot and going to the charger is equal to the procedure for going to the sector to clean, but only with another node as the end goal. When the route is completed, the node list and the sector to clean are handed over to the agent who starts the route.

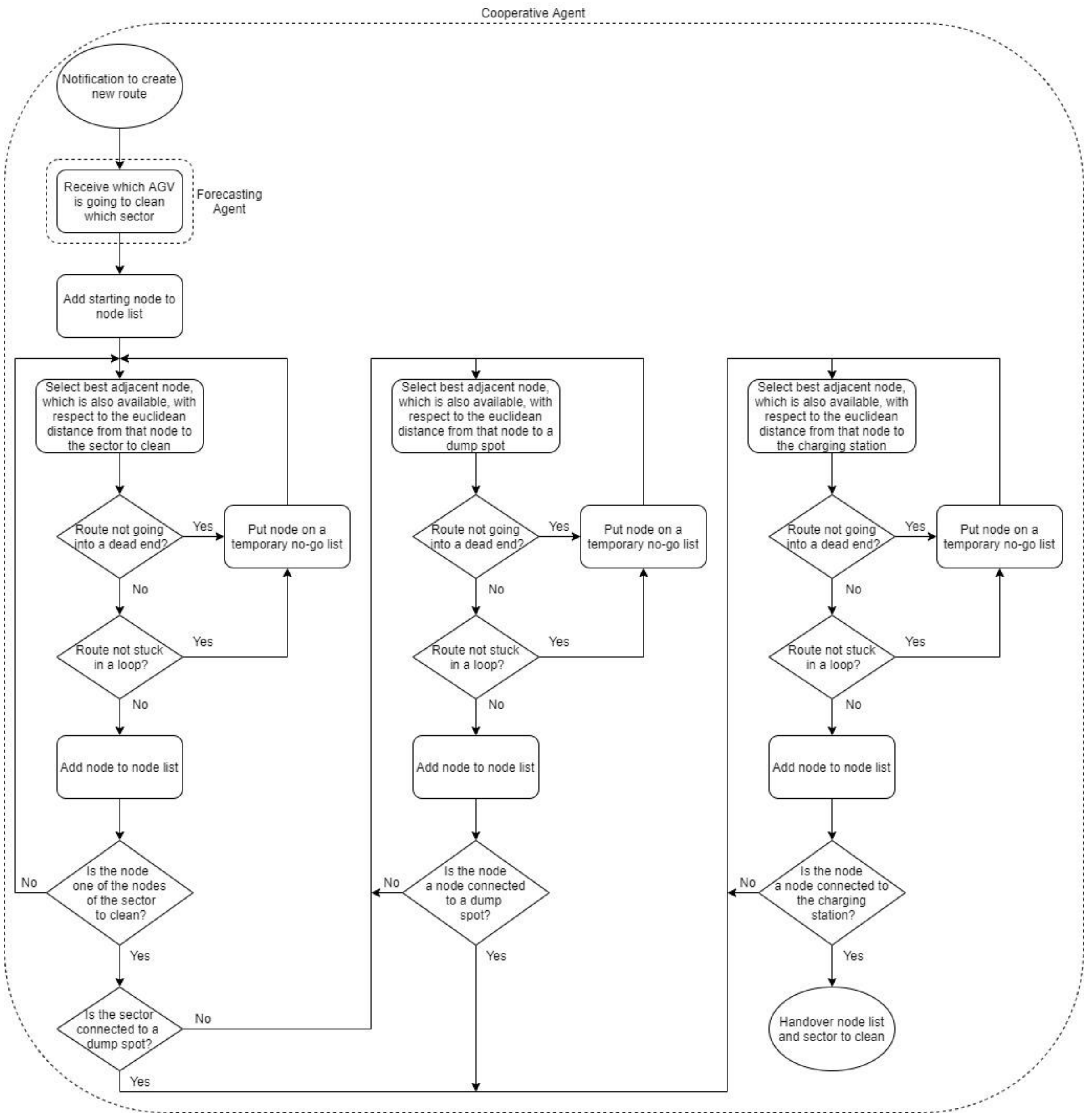

Cooperative Agent
Notification to create new route
Receive which AGV is going to clean which sector
Forecasting Agent
Add starting node to node list
Select best adjacent node, which is also available, with respect to the euclidean distance from that node to the sector to clean
Route not going into a dead end?
Yes
Put node on a temporary no-go list
No
Route not stuck in a loop?
Yes
No
Add node to node list
Is the node one of the nodes of the sector to clean?
No
Yes
Is the sector connected to a dump spot?
No
Yes
Select best adjacent node, which is also available, with respect to the euclidean distance from that node to a dump spot
Route not going into a dead end?
Put node on a temporary no-go list
Route not stuck in a loop?
Add node to node list
Is the node a node connected to a dump spot?
Select best adjacent node, which is also available, with respect to the euclidean distance from that node to the charging station
Route not going into a dead end?
Put node on a temporary no-go list
Route not stuck in a loop?
Add node to node list
Is the node a node connected to the charging station?
Handover node list and sector to clean

Figure 4-2: Flowchart new route heuristic

Extending this heuristic to multiple routing strategies is easy because the primary input of making the routes are the nodes of the network. The first extension is the strategy where the more giant arcs are split up into multiple smaller ones. In this case, it becomes possible to clean an arc where a failure occurred partly instead of blocking and not cleaning the whole arc. This research also gives insight into the improvements if multiple smaller adjacent arcs, which are in one line, are combined. This extension is easily done in the heuristic by chancing the endpoints of an arc. Another strategy this research gives insights on is the option to block specific arcs for specific AGVs. This option can positively impact the traveling distance if arcs are blocked on the opposite side of the barn or can prioritize specific arcs over other arcs.

4.5.2. Adapt route heuristic

The second heuristic for routing the AGVs is for adapting the route during a route when needed. This Heuristic is also based on the NNH. This heuristic's capacity constraints are hard to control because it is called in the middle of a route. Therefore the AGV is sent as fast as possible to a dump spot or the charger to minimize the chance of a tank overflow or empty battery. This heuristic is needed because of the shared lanes and facilities. Because this is shared, two AGVs can come across each other unexpectedly. The big difference between the optimization heuristic and this heuristic is that the optimization heuristic is making and planning entirely new routes, whereas the adapt route heuristic only handles unexpected situations. This case occurs when an AGV comes across another AGV with a failure during its route. When they are in a specific range of each other, the chance of a collision increases. This increase is partly due to the inaccuracy of the SMART heuristic.

The procedure is triggered by a notification of a failure within a specific range of the AGV who notifies the failure. The heuristic starts by setting the end destination (Figure 4-3). The destination is the charger if the AGV has already been to the dump spot during the current route. If not, the AGV is going first to a dump spot. The second step is to delete the current route out of the system at the AGV so a new one can be made. The next step in the procedure is checking in the neighborhood (within a couple of meters) what the best next place to go is. This procedure continues until the destination is reached. Following this procedure, it can be possible that the AGV is guided around the other AGVs with a failure. All previous locations are saved in a tabu list to avoid the AGV getting stuck in a dead-end or a loop.

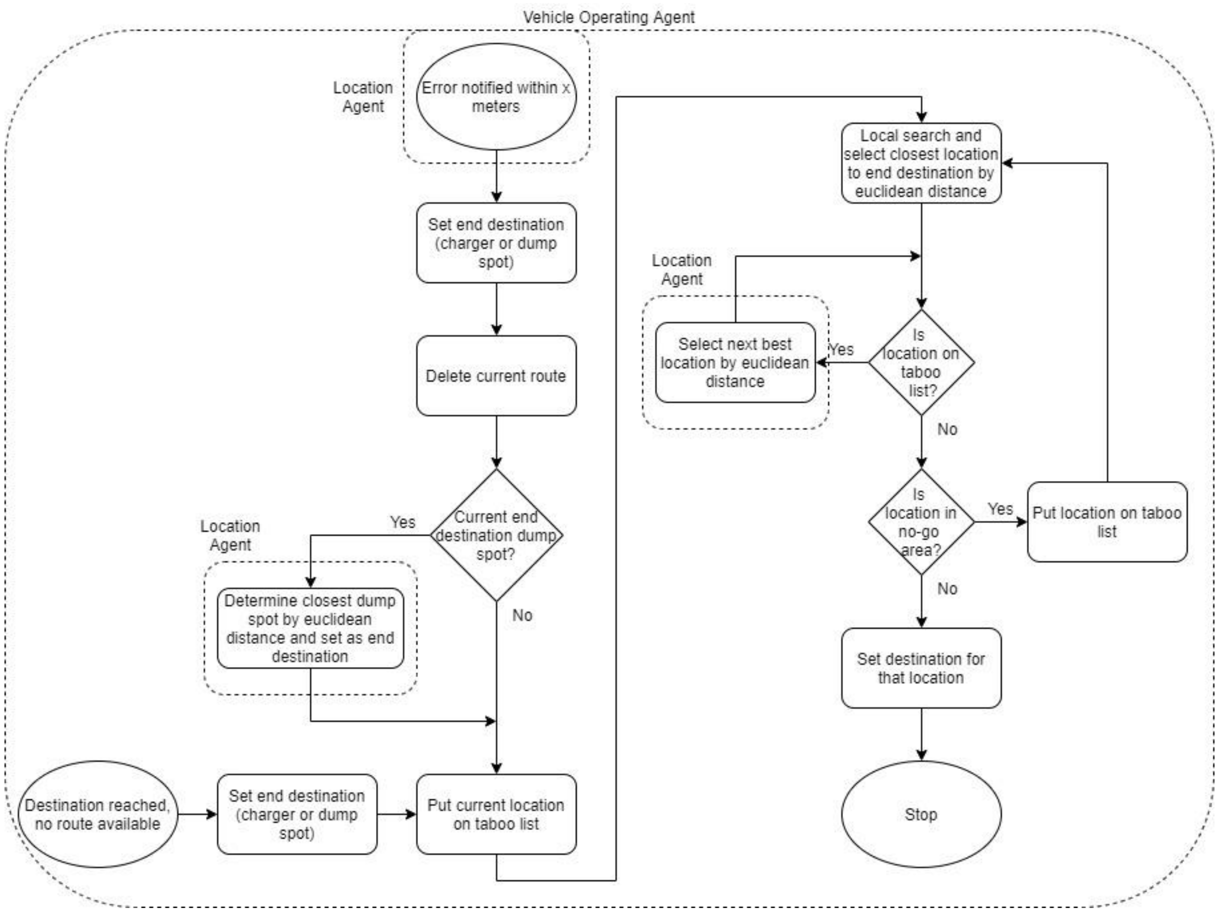

Vehicle Operating Agent
Location Agent
Error notified within x meters
Set end destination (charger or dump spot)
Delete current route
Current end destination dump spot?
Yes
No
Location Agent
Determine closest dump spot by euclidean distance and set as end destination
Destination reached, no route available
Set end destination (charger or dump spot)
Put current location on taboo list
Local search and select closest location to end destination by euclidean distance
Location Agent
Select next best location by euclidean distance
Is location on taboo list?
Yes
No
Is location in no-go area?
Yes
No
Put location on taboo list
Set destination for that location
Stop

Figure 4-3: Flowchart route adaption

4.6. Conclusion

In this chapter, we have introduced the use case that is used for the simulation. The barn is a testing barn of Lely, which contains two Lely Discovery's 120 robots and two Lely Astronaut A4 AMS. The maximum capacity of the barn is 120 cows.

In this use case, we implement three scenarios. The first two are implemented at the same time. These are the shared facilities and the task handovers. The optimization heuristic determines which task is handed over and which facilities are used at what time. This heuristic is based on the NNH, taking CARP constraints into account. The heuristic makes sure that the next route to determine covers the most urgent sector to visit.

The third scenario is the fully autonomous scenario where the AGV's do not have any pre-programmed routes anymore, and the routes are only determined with the optimization heuristic. This route covers the most urgent sector to visit.

5. Conceptual model

This chapter describes the conceptual model and partly answers the sub-questing: *How would a simulation model look like for a self-organizing system for barn cleaning robots?* A conceptual model is a framework used as a bridge between a problem situation and a simulation model. In this framework, all components that are needed to build a simulation model are identified. A conceptual model is always made as part of a simulation study and not in isolation. Because of that, it is possible that both the conceptual model and simulation model can be adapted during the design of one of them. We use the framework from Robinson (Figure 5-1) to design our conceptual model, a commonly used framework for simulation models. This framework consists out of five stages:

- Understanding the problem situation
- Determining the modeling and general project objectives
- Identifying the model outputs (responses)
- Identifying the model inputs (experimental factors)
- Determining the model content (scope and level of detail) and identifying any assumptions and simplifications

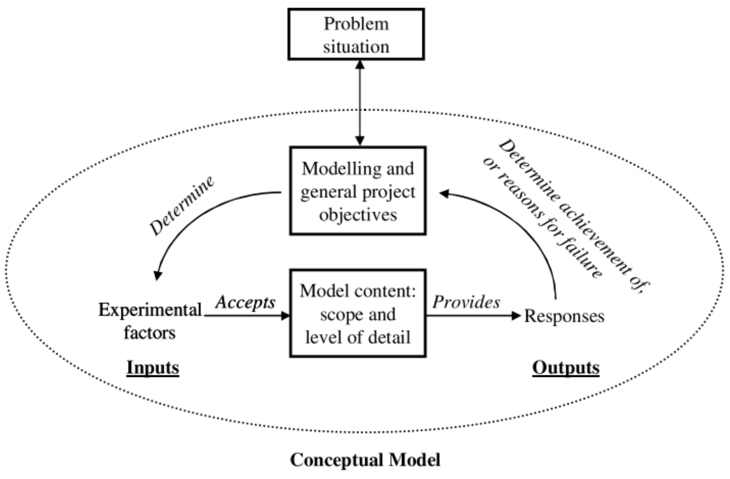

Problem situation
Modelling and general project objectives
Determine
Determine achievement of, or reasons for failure
Experimental factors
Accepts
Model content: scope and level of detail
Provides
Responses
Inputs
Outputs
Conceptual Model

Figure 5-1: Conceptual model framework (Robinson, 2008)

5.1. Problem situation

This section discusses the problem that the simulation model faces, which is the first step of the framework. In previous chapters, we have discussed the problem context extensively. In this section, we are going to link this to the conceptual model.

Firstly, the simulation model for the Lely barn cleaning robot case should provide enough accurate insights on the best routing strategy when a failure is detected in the environment and the best communication strategy between the different agents. Secondly, the model must be (visual) detailed and clear enough to have the trust and confidence of all the consortium partners to trust the outcomes. Thirdly, the model should be feasible to build within data and time constraints. And at last, the model must be constructed so that it can apply to other use case environments (different barns) and be extendable for other scenarios.

5.2. Modeling and general project objectives

The organizational aim of Lely is to have as good as possible responses of other cleaning robots when one of the cleaning robots gets a failure in the environment to create a clean and healthy as possible environment for the cows. To determine what the best approach is for tackling this problem, we need our modeling objectives:

- To determine the best routing strategy.
- To determine the best communication strategy.

We aim to design a flexible and generic model where components are reusable in other models. Also, the model needs to be easy to adapt and extended for other scenarios. Because we have many different experiments that we have to run, we pay extra attention to the level of detail to keep the total run time within reasonable bounds. To keep the model easy to understand, we make the model with 3D animation, show real-time data updates during simulation runs, and make change-logs in the code.

5.3. Model outputs

In this section, we introduce the model outputs that we want to have. These outputs are based on the KPIs, which are determined together with the consortium. The output data is generated with our simulation model. This data is also the data on where we are going to base our conclusions. The main KPIs be:

- The average quantity of manure per area per time unit. This average is measured over the whole barn and per sector.
- The squared average quantity of manure per area per time unit. Because the manure production is uniform, the squared value of the first KPI tells us how long manure is laying on an area section.
- A histogram with all quantities of picked-up manure per area with intervals of 0.25 liters.
- A histogram with all times since the last visit per area with intervals of 15 minutes.

Next to these four main KPIs, we also recover other data from the model to analyze:

- A list with all the failures that occurred with the start and end time, location, current route, and robot.
- A list with the number of times a standard route is driven by a robot and the number of times a robot had driven an additional route.
- A list with all route data per route. This list contains all the charging time, dumping time, driving time, and the quantity of manure picked up and dumped manure.

5.4. Model inputs

This section introduces all the model inputs we need to come to a proper simulation model. The first set of model inputs are the experimental factors:

- Cleaning robot to cleaning robot communication range (communication strategy)
- Cleaning robot to central system communication range (communication strategy)
- Prioritizing sectors (routing strategy)
- Number of arcs (routing strategy)
- Using fixed route and schedule (routing strategy)

Next to the experimental factors, we need other model input as well. These model inputs are received from Lely or the route analysis.

- The manure production per cow per day
- The number of cows in the barn

- The blueprint of the barn with all dimensions concerning cubicles, feeding spots, AMS, and charging station
- The number of cleaning robots in the barn
- The dimensions of the cleaning robot
- The capacity of the robot (battery and tank)
- The driving speed of the cleaning robot
- The time needed the dump manure
- The energy consumption when cleaning
- The charging speed
- The maximum driving/cleaning time on a full battery
- The failure interval of a robot
- The failure duration of a failure during night time and during day time
- The current routes with schedule
- The shortest distance two robots can get from each other
- Routing strategy
- Conflict avoiding strategy

5.5. Model content

In this section, we discuss the scope and level of detail the simulation model has. We first discuss the cows and manure production. We assume that all the cows in the barn are the same and do not change over time, so we have a homogenous herd of cows. We also assume that the manure production is uniform over the day. For simplicity, we also assume that manure production is divided into three areas: high intensity, average, and low intensity. Which area is which level of intensity is depending on the barn layout and function of the area.

We assume that the cleaning robot continuously drives with the same speed and has this speed directly from the start. For model simplicity, we assume that the robot does not have to back up before going forward again for the cornering and docking procedures. When the robot starts after a failure, it finishes the route driven when the failure occurred. The duration of failures follows a specific probability distribution, which differs when a failure occurs during the nighttime or the daytime. The interval between two failures also follows a specific probability distribution, which can be found in Appendix P. The failure rates between different robots are independent. The tables in Appendix M give a more detailed overview of what is included and excluded in the simulation model.

5.6. Process flow

The last phase in the conceptual model is to illustrate the process flow. These illustrations consist of logic flowcharts of how the simulation model responds to events. We define the different processes per agent.

Vehicle Operating Agent

The Vehicle Operating Agent is responsible for all the basic tasks. An essential task is starting the robot and let it drive its route. The first logic flowchart is for the schedule checking process (Figure 5-2). This procedure checks after each time unit if a route is scheduled to start in the time frame between the current and last time unit. It first checks if the scheduled AGV does not have a failure. If so, it put the AGV in failure. If the AGV does not have a failure, the procedure checks if there is a route to start and what kind of route. If so, check if the AGV is not already driving a route and if the battery level is high enough to finish the next scheduled route. If everything is correct, the procedure triggers another event to start the AGV.

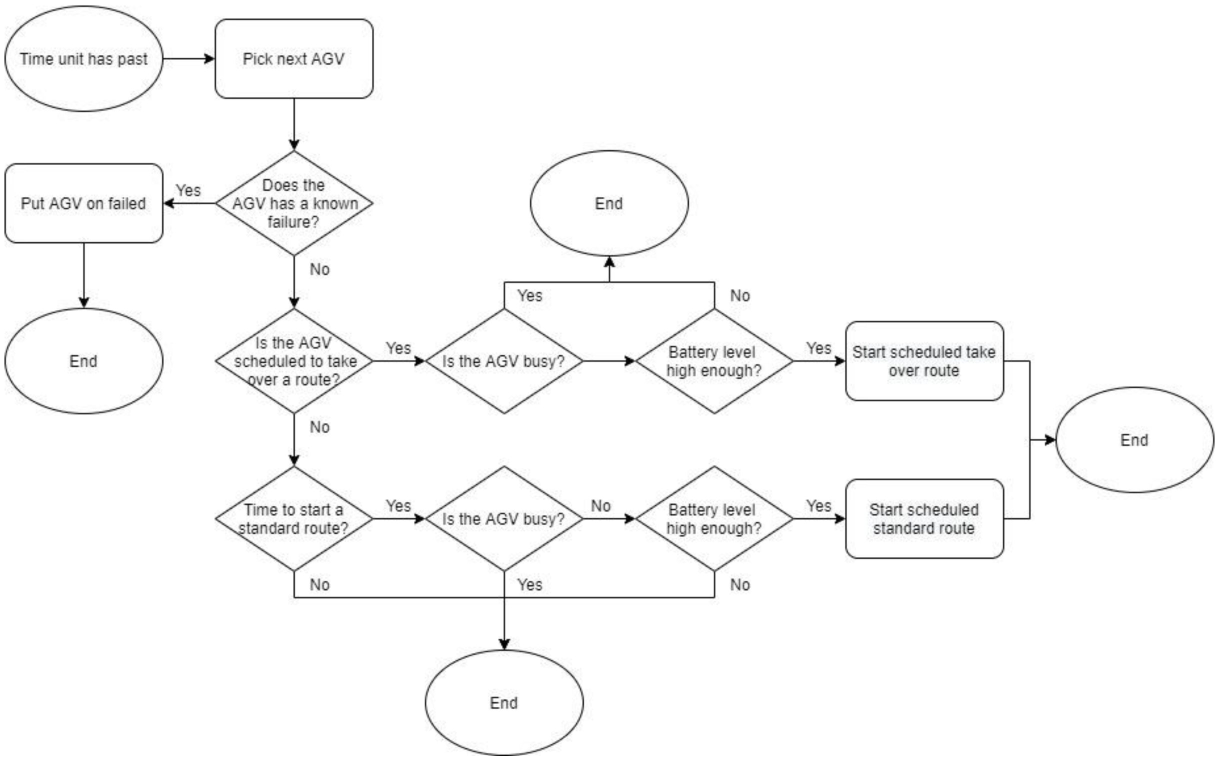

Time unit has past
Pick next AGV
Does the AGV has a known failure?
Yes
Put AGV on failed
End
No
Is the AGV scheduled to take over a route?
Yes
Is the AGV busy?
Yes
End
No
Battery level high enough?
Yes
Start scheduled take over route
End
No
Time to start a standard route?
Yes
Is the AGV busy?
No
Battery level high enough?
Yes
Start scheduled standard route
No
Yes
No
End

Figure 5-2: Logic flowchart schedule checking

The next event where the Vehicle Operating Agent is responsible for is starting and driving the next route (Figure 5-3). This procedure is a straightforward procedure where only the end destinations (dump spots and charger) are set, and an array with turning points is loaded on the AGV.

The last two events where the Vehicle Operating Agent is responsible for are pretty similar to each other. These are the events where an AGV tries to communicate with another AGV, and the other event is where an AGV tries to communicate with the central system. Both events are triggered when a time unit has passed.

When trying to communicate with another AGV, the AGV searches in a specific range for another AGV (Figure 5-4). If there is one, they both hand over information about their status. If one of the AGVs is failed, there is a check if the AGVs are not getting too close to each other. When that is the case, another event is activated to avoid a possible collision.

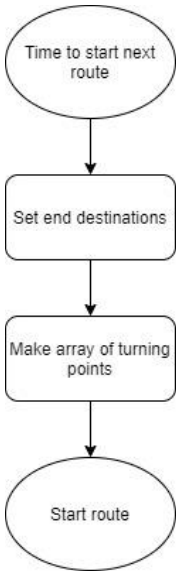


Figure 5-3: Logic flowchart next route

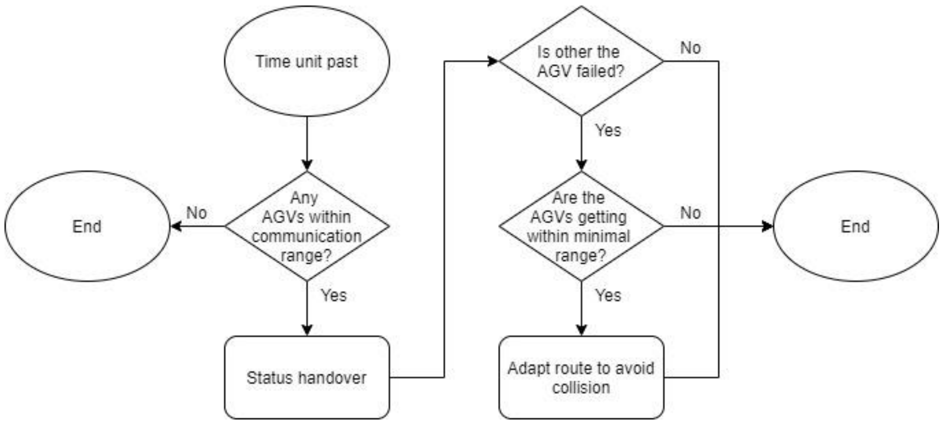

Time unit past
Any AGVs within communication range?
No
End
Yes
Status handover
Is other the AGV failed?
No
Yes
Are the AGVs getting within minimal range?
No
End
Yes
Adapt route to avoid collision

Figure 5-4: Logic flowchart AGV to AGV communication

Trying to communicate with the central system (Figure 5-5) starts the same as the event trying to communicate with another AGV. Different in this one is that not only the own status is handed over, but also the other AGV(s) status. For all AGVs, the status is checked if there are unknown failures. If so, a failure message is sent. In the end, the procedure updates the list with failures.

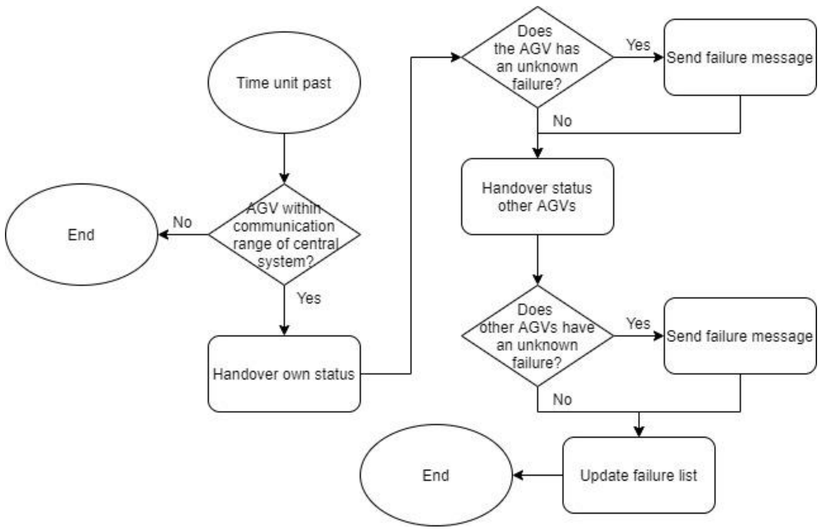

Time unit past
AGV within communication range of central system?
No
End
Yes
Handover own status
Does the AGV has an unknown failure?
Yes
Send failure message
No
Handover status other AGVs
Does other AGVs have an unknown failure?
Yes
Send failure message
No
Update failure list
End

Figure 5-5: Logic flowchart AGV to central system communication

Cooperative Agent

The cooperative agent is responsible for all the advanced tasks when the system is interrupted during the standard procedures and schedules.

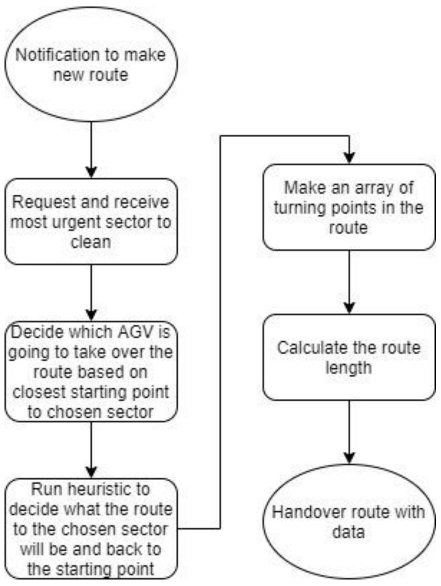

Notification to make new route
Request and receive most urgent sector to clean
Decide which AGV is going to take over the route based on closest starting point to chosen sector
Run heuristic to decide what the route to the chosen sector will be and back to the starting point
Make an array of turning points in the route
Calculate the route length
Handover route with data

Figure 5-6: Logic flowchart new route

There are two critical events: a new route is created, and the schedule is adapted to schedule that new route. New routes are created when there is a request for one. The process starts with a request to the Forecasting Agent what the most urgent sector to clean is. This choice is made based on the time of the last visit. The next step is to decide which AGV is going to clean the most urgent sector to clean. This decision is made by selecting the AGV, which starting point is the closest to the sector. The next step is to run the optimization heuristic discussed in the previous chapter, making the entire route. When this is done, the route array is made, and the length of the route is calculated. This data eventually be handed over to the vehicle operating agent for starting and driving the route.

When the data is handed over, the adapt schedule event is triggered (Figure 5-7). This process temporarily blocks the assigned AGV's standard schedule and adds the new route into the schedule with the correct information.

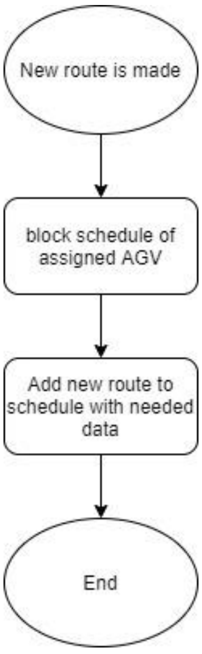

New route is made
block schedule of assigned AGV
Add new route to schedule with needed data
End

Figure 5-7: Logic flowchart adapt schedule

Forecasting Agent

The Forecasting Agent is responsible for making all the forecasts in the environment. In this simulation, the forecast is only a basic forecast based on the time of the last visit of every area. This information is coming from the Monitoring and Logging Agent.

The process starts with determining all times of the last visit for all the areas (Figure 5-8). Then the sector with the most urgent areas is picked as the most urgent sector to clean. This information is handed over for creating a new route.

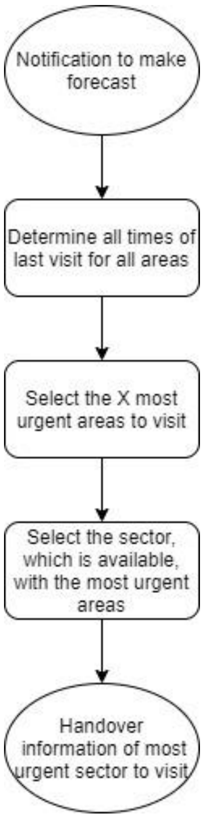

Notification to make forecast
Determine all times of last visit for all areas
Select the X most urgent areas to visit
Select the sector, which is available, with the most urgent areas
Handover information of most urgent sector to visit

Figure 5-8: Logic flowchart forecasting

Monitoring and Logging Agent

The Monitoring and Logging Agent is responsible for logging data and monitoring the data and environment. This agent is responsible for one event in the model, the status check of the sectors (Figure 5-9**Error! Reference source not found.**). This event is triggered by a request to update the status of all sectors. The first step is to reset the current list. After that, the procedure checks which sectors have an area that is blocked due to a failure. If a sector has an area blocked, the whole sector is blocked for driving and cleaning. So it cannot become a part of a new route that be created due to the failure.

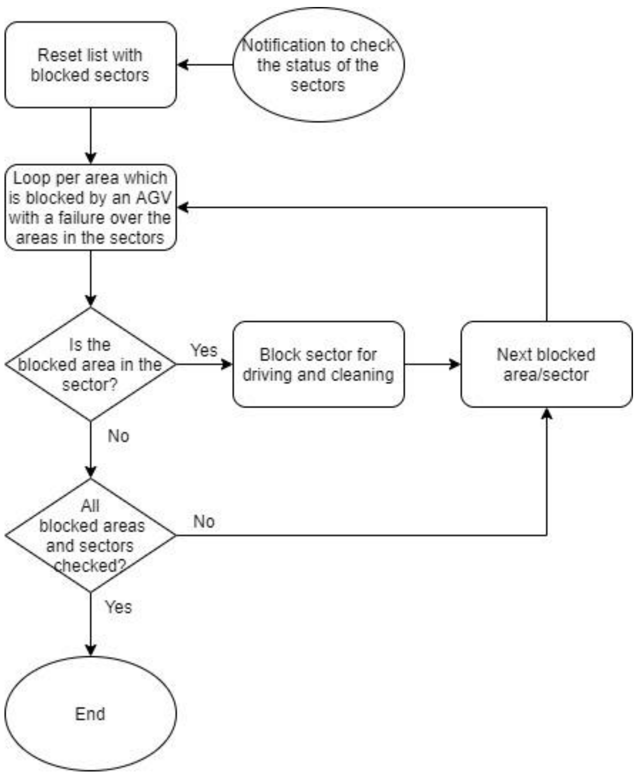

Notification to check the status of the sectors
Reset list with blocked sectors
Loop per area which is blocked by an AGV with a failure over the areas in the sectors
Is the blocked area in the sector?
Yes
Block sector for driving and cleaning
Next blocked area/sector
No
All blocked areas and sectors checked?
No
Yes
End

Figure 5-9: Logic flowchart sector check

Vehicle Status Agent

The Vehicle Status Agent is responsible for keeping track of the remaining capacities of the AGV; this can be loading capacity and the battery level. In this model, the only job of this agent is to hand over the battery level when the Vehicle Operating Agent is checking if it is possible to start a route.

Location Agent

The Location Agent is responsible for starting two events, the route adaption when an AGV comes too close to another AGV and the area blocking.

The route adaption process starts with checking if the charger is the next destination (Figure 5-11). If not so, the AGV first has to go to the dump spot to dump its manure. A heuristic discussed in the previous chapter handles how the AGV drives to one of these destinations.

The area blocking process (Figure 5-10) is triggered when an AGV comes across an AGV with a failure that was not known yet. The process starts with determining the location of the AGV with a failure. The next step is to check which areas are in the range of that location. If an area is in that range, it is blocked.

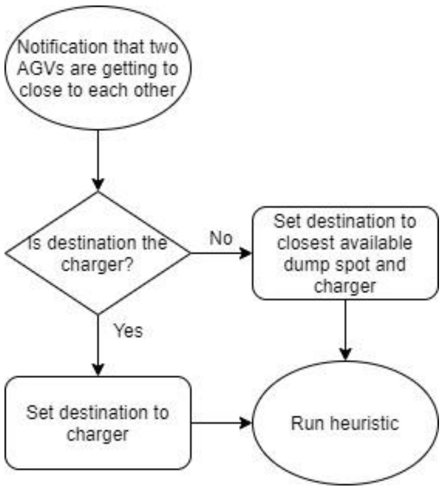

Notification that two AGVs are getting to close to each other
Is destination the charger?
No
Set destination to closest available dump spot and charger
Yes
Set destination to charger
Run heuristic

Figure 5-11: Logic flowchart route adaption

5.7. Conclusion

In this chapter, we have designed a conceptual model based on the framework of Robinson (2008) for building a simulation model of our use case with the implemented MAS and heuristics. It became clear that it is essential to vary different routing approaches to see what the best approach is to implement in the existing system of Lely. Furthermore, it is essential to vary in the different communication strategies to see how the system performs in different environments where communication between the different entities can be hard to guarantee.

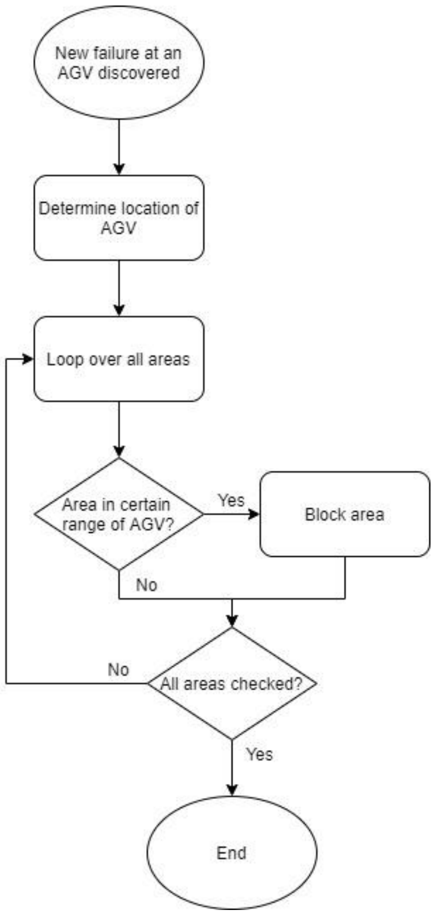

New failure at an AGV discovered
Determine location of AGV
Loop over all areas
Area in certain range of AGV?
Yes
Block area
No
No
All areas checked?
Yes
End

Figure 5-10: Logic flowchart area blocking

6. Simulation

In this chapter, we describe our implemented simulation model. The model is made in Siemens Plant Simulation version 15.2. In the simulation model, we implement and test the hybrid approach and the central approach of the MAS. We start by giving a technical description of the simulation model we are using for the simulation study. Secondly, we describe the input data we have collected and used in the simulation model. Thirdly, we are going to describe our experimental design. After that, we describe the sensitivity analysis we are going to perform. In the fifth part, we show how we validate our model. At last, we discuss the simulation setup to determine the warm-up length, run length, and the number of replications. Together, we can partly answer the sub-questing: *How would a simulation model look like for a self-organizing system for barn cleaning robots?*

6.1. Model description

In this section, we explain the simulation model that we have used in the simulation study. In Appendix N, we discuss the model more in-depth. In Plant Simulation, the program we are using, the code is split up into small pieces of codes in so-called methods with each their purpose and goal. We have recreated the barn with 3D objects for better visualization (Figure 6-1). On the floor of the barn, we have laid out a grid with markers. Each of these markers represents a small piece of the area of the barn floor. These markers hold data of that piece of floor. The manure is visualized on the markers with a color, white for a clean floor and black for a dirty floor.

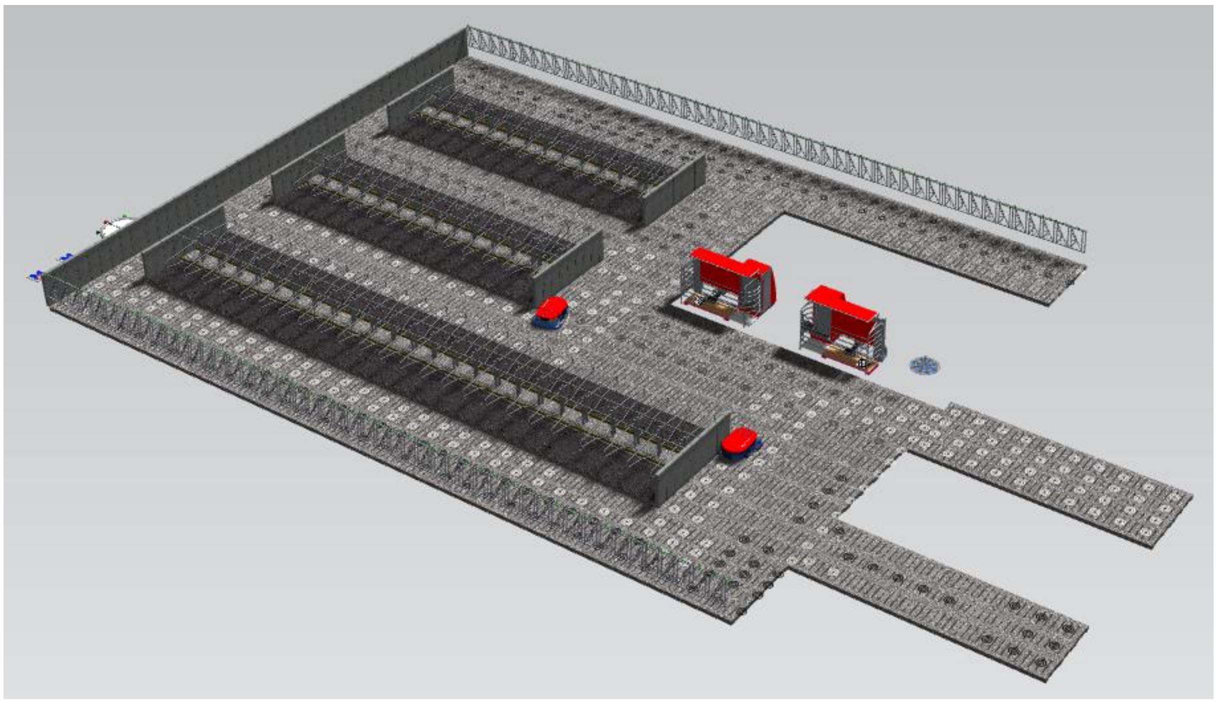


Figure 6-1: Screenshot of the 3D simulation model

6.2. Input data

In this section, we discuss the analysis of the input data. In Section 5.4, we already discussed all the input data we need for the simulation model. We have received this input data from Lely of the use case barn. But some of this data needs to be transformed into good input data for the simulation model.

We have received one day of log data of the driven routes of both cleaning robots from the use case barn. This data is analyzed. In total, seven different preprogrammed routes are driven by the two robots. The most critical data from this analysis can be seen in Table 6-1. A more extensive analysis can be found in Appendix A.

Table 6-1: Route analysis

Route	# analyzed	# normal	AvgDrivingTime	AvgTotalDumpTime	AvgEffecitveDrivingTime
r1r1	12	12	00:21:02	00:01:00	00:20:02
r1r2	12	12	00:17:06	00:01:00	00:16:05
r1r3	6	6	00:15:44	00:01:00	00:14:44
r1r4	5	6	00:26:12	00:01:00	00:25:12
r2r1	6	6	00:17:29	00:00:30	00:16:59
r2r2	10	12	00:35:19	00:01:30	00:33:50
r2r3	5	6	00:29:39	00:00:30	00:29:09

Considering the failures that occurred that day and setting the same time interval between the same routes, we conclude that four routes are driven every four hours, and three routes are driven every two hours.

Knowing the routes through the barn and the number of times each route is driven per day gives us information on which parts of the barn need to be cleaned more frequently than other parts. To visualize this, we have created a simplified map of the barn with area blocks of around $1m^2$. By counting the times a robot passes each block, we can create a heatmap (Figure 6-2) that shows us the barn areas that are cleaned more frequently than other areas.

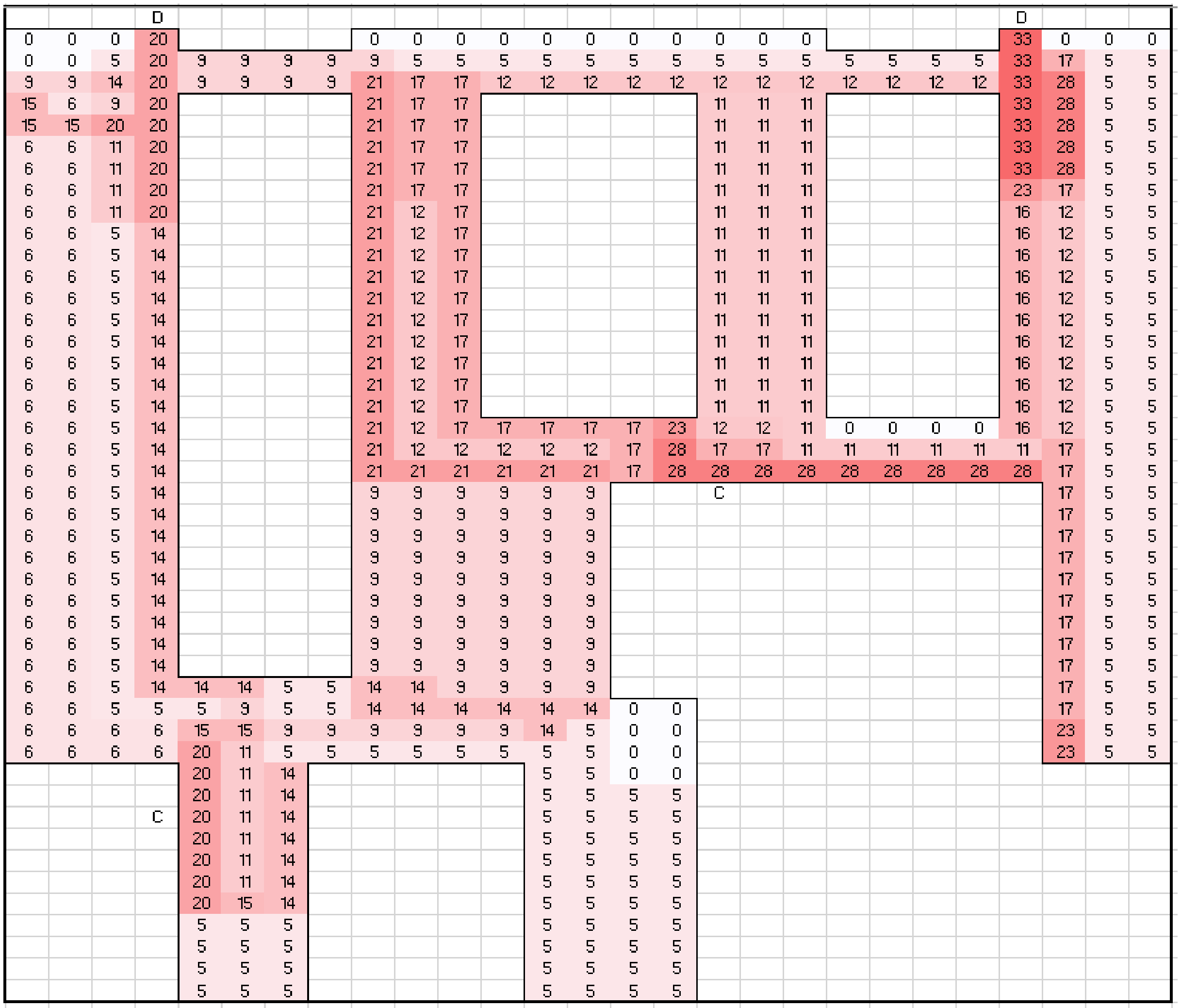


Figure 6-2: Barn heatmap

This heatmap (Figure 6-2) shows that some parts are more often cleaned than other areas. Areas with the highest frequency are around dumping spots (D) and charging stations (C). These high frequencies make sense as the robots are driving around those places every route. Also, we see that the robots are cleaning more often than average next to the cubicles. Here are the cows more often at one specific spot for a more extended period and therefore discharge their manure in that specific place. We see the opposite close to the feeding fence because the cows stand only head forward at this place.

At last, we see two parts at the downside of the heatmap that does not need much cleaning. These parts of the barn have a low intensity of cow traffic due to the layout of this specific barn. This barn has been renovated and went from a conventional milking stand to an AMS. The former milking stand was in that area and is now an area where the cows can only walk.

For the failure length and interval, limited data is available. The received log data from Lely was too limited to get a proper distribution over failure length and interval. The information we have received from Lely about the failures is that the average interval between two failures is 2.5 days, and the length of a failure during day time is 30 minutes till 3 hours and up to 8 hours during the night. The input used during the simulation for the interval between failures is a normal distribution with a mean of 2.5 days with a lower bound of 1 day and an upper bound of 5 days. For the failure length distribution, we have followed the calculations from Law (2015). He suggests determining the distribution in a lack or absence of data to follow a Weibull distribution. In our case, we come to a Weibull distribution for the daytime with a $\alpha = 5$, and a $\beta = 7528.6$ with a lower bound of 30 minutes and a higher bound of 1 day. The distribution for the nighttime is similar, only the $\beta = 20076.3$. The calculations of this can be found in Appendix P.

6.3. Experimental design

This simulation study is split up into four experiments. The first two are to test and validate the model and see the impact of robot failures in the system. In those two experiments, we are not introducing anything new compared to the current situation. In the third experiment, we introduce our MAS with the heuristics explained in Section 4.4 combined with the standard routing schedule, task handovers, and shared facilities. In the fourth and last experiment, we use heuristics to create routes without any pre-programmed routes. Table 6-2 summarizes each experiment with the case explanation and goal.

Table 6-2: Experiments

Experiment	Case	Goal
1	Current situation without robot failures	To validate the simulation model with the real world
2	Current situation with robot failures	To see the current impact of robot failures in the system
3	Situation with robot failures, task handovers, shared facilities (scenario 1 & 2), and MAS implementation	To see the performance of the different routing strategies and the finding the best approach for the MAS
4	Situation with a fully self-organizing system without pre-programmed routes (scenario 3)	To see the performance of introducing a fully self-organizing system

The MAS is introduced in experiments three and four. In the case study, we want to test the performance of this system with different communication configurations. The difference in configurations goes from only communication possibilities between the robots and the central system when a robot is docked to a charger, a wireless connection in several areas in the barn, and a continuous connection between robots and the central system. The values that are going to be used can be found in Table 6-3. The connection points to the central system are the most logical points in practice to have internet access. These are the charging points, the AMS, and the barn sides, where cable connections often come into the barn.

Table 6-3: Experimental factors

Type of connection	Initial value	2nd value	3rd value
Robot to robot	10m	15m	20m
Robot to central system	1m (only at the charger)	10m	20m
Priority sectors	false	true	-
Number of arcs/sectors	Small	large	-

6.4. Sensitivity analysis

We test the influence of different input variables for the sensitivity analysis and how the system responds to them. The first variable is the number of cows that are living in the barn. If the number of cows changes in the barn, then the total manure production is also changing. We increase the number of cows to see if the system can handle it concerning capacity. Also, we test the influence of the number of failures, combined with the length of failures, of the robots. In that way, we are testing what happens if the conditions in the barn are getting worse and what the influence is if the MTTR is shorter due to the faster response of the system owner. The values that we are going to use for this can be found in Table 6-4.

Table 6-4: Sensitivity analysis factors

Analysis on	**Initial value**	**First value**	**Second value**
Number of cows	120	135	150
Failure duration day	$\beta = 7528.605$	$\beta = 4391.686$	-
Failure duration night	$\beta = 20076.279$	$\beta = 13175.058$	-
Failure interval	2.5 days on average	1.25 days on average	-

6.5. Model validation and verification

We have validated our model to see how representative the simulation model is compared with the actual situation. This validation has been done in multiple ways. The first one is running the model for one day of simulation time without generating failures and comparing the basic route statistics with the route analysis on the data received from Lely. The results of this run can be found in Table 6-5.

Table 6-5: Validation run results

Route	#	Average driving time	Average effective driving time	Average total effective driving time
r1r1	12	00:21:57	00:20:54	00:20:54
r1r2	12	00:16:21	00:15:18	00:15:18
r1r3	6	00:15:42	00:14:39	00:14:39
r1r4	6	00:23:00	00:21:57	00:21:57
r2r1	6	00:14:34	00:14:02	00:14:02
r2r2	12	00:40:26	00:38:52	00:38:52
r2r3	6	00:33:17	00:32:46	00:32:46

Compared with the results of Table 6-1, we can see some minor differences. These differences are because taking corners and the docking procedures for dumping and charging are hard to equal and realistically reproduce in the simulation model. To produce more realistic results, we have decided to multiply the speed of the AGV's with a factor. This factor is 1.056 and is the weighted average of the time difference of the different routes and the times the routes are driven generally on a day. The calculations of this factor can be found in Appendix Q.

Furthermore, we have programmed our model part by part. Before finishing and starting coding on the next part, we debug and test the part we just had finished to see if it works according to expected. At last, we have done a visual verification to see if the AGV's are having normal realistic behavior during the different operations they can perform.

6.6. Simulation setup

In this section, we discuss how we set up the simulation. We show the calculations of the warm-up period, run length, and the number of replications. These values are determined using the initial and average manure per marker per time unit as the primary KPI.

6.6.1. Warm-up period

The warm-up period is the period of time the simulation model needs to get in a steady state. As soon as the simulation is past this period, the data is valid and can be used for analysis. Because the simulation we are going to run is non-terminating, we have to use a warm-up period. To determine the warm-up period, we have used Welch's graphical method. We used five replications to determine the warm-up length for each experiment and took the moving averages with different window sizes over the mean of every observation. The graph in Appendix R shows that the model is steady-state after around 300 minutes, equal to 5 hours. Next to that, have we chosen to use the replication/deletion approach.

6.6.2. Run-length

The run length is the time the simulation is running and collecting data. This length must be long enough to make good conclusions and not too long because of the computation time. The standard rule of thumb is that the run length is ten times as long as the warm-up period. In our case, that would mean that the run length has to be 50 hours, which is just over two days. Because the main point of interest in our simulation study is the failures of the AGV's and those only occur every 2.5 days on average, we have decided to expand the run length to six days. This decision is based on a test run and a quick analysis of the data to see how often a failure occurs in the system.

6.6.3. Number of replications

To reduce the chance of coincident and prove that the outcome of the data is statistically significant with a 95% confidence interval, we are running every experiment multiple times with the same input values but with different common random numbers (CRN) every time. To determine the number of replications, we run *n* independent replications with length *m*, where *m* is much larger than the warm-up period. We use the formula $\frac{t_{n-1,1-\alpha/2}\sqrt{S^2/n}}{\bar{X}} < \gamma'$ where $\gamma' = \frac{\gamma}{1-\gamma}$ eventually to set the number of replications (Law, 2015). We do not need multiple replications for the first experiment because the stochastic part is not introduced yet. We need at least three replications in the second experiment, and in the third and fourth experiment, we need four replications. The calculations for this can be found in Appendix R. For simplicity in modeling and to compare the different experiments with each other, we have set the number of replications for both experiments to four.

6.7. Conclusion

This chapter explains how the simulation model and the experiments we run in this model look. The model has been validated and corrected by comparing the run data with the actual data received from Lely.

The first experiment that is conducted is equal to the current situation without any failures in the system. This experiment is conducted to set a benchmark for the other experiments. The second experiment is the same as the first experiment but with occurring failures at the AGV's. Comparing this result with experiment one, we can see the impact of the failures on the system. The third experiment is split up into three parts. In all parts, we are introducing the scenarios of shared facilities and task handovers. In the first part of this experiment, we do not make use of priority areas. In the second, we make use of these. In the last part, we split the larger arcs into multiple smaller ones to see if splitting up is beneficial.

In the fourth and last experiment, the third scenario introduced where the AGV's does not have any preprogrammed routes anymore but are only driving routes made with the heuristic. In this experiment, we also check if it is beneficial to connect adjacent arcs and the influence of giving ownership of an arc to specific AGVs.

In the experiments, we change the communication ranges for AGV to AGV and AGV to the central system to see the influence of the level of communication on the system's performance. In total, we replicate every experiment configuration four times for six days with a warmup period of five hours. To see how the system reacts when the environment changes, we are also conducting a sensitivity analysis. In this analysis, we check the performances when the number of cows is increased in the barn. For the failures, we check the performances when the influence of the failure length and the failure interval is reduced.

7. Results

In this chapter, we discuss the results of our simulation study and answer the sub-question: *How is a self-organizing control system for barn cleaning robots performing?* We start by analyzing the first two experiments. By comparing the outcome of those two experiments, we have insights into the impact of the failures in the system. Then we are going to discuss the results of the impact of the different communication ranges and configurations. As third, we discuss the impact of the different routing strategies based on the best communication strategy. After that, we discuss the results of the simulation study with a fully autonomous planning system. At last, we are going to discuss the results of the sensitivity analysis.

7.1. Impact of failures

In this section, we discuss the impact of the failures of the AGV's in the system. We discuss the impact by comparing the primary KPI, the average quantity of manure per marker, of the first two experiments with each other. This result is used as a benchmark for the other experiments to see how they are performing.

We performed the first experiment without failures. As expected does this experiment have repeating cycles for the KPIs average quantity per area and the penalty cycle, which can be seen in Figure 7-1. These repeating cycles are because the manure production is uniform, the AGV's are driving according to a fixed schedule, and there are no interruptions in the system, which affects this KPI.

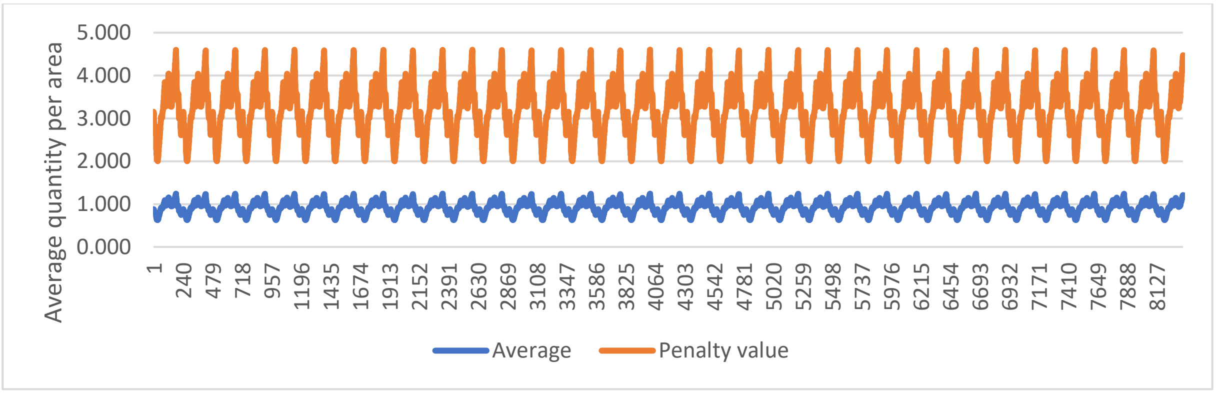


Figure 7-1: Results experiment 1

The first experiment concludes that the average quantity of manure per area is 0.921, and the average penalty value is 3.208 measured over six days.

The results of experiment two, where the failures are introduced in the system, are logically performing worse than experiment one. Compared with experiment one, we can say that the failures let the average quantity of manure per marker increase by 3.66% to 0.955 and the penalty value by 6.66% to 3.422. This increase is caused by 3.5 failures in the system consisting of two AGV's measured over six days. This result is also the benchmark for other experiments.

Analyzing the histograms with all quantities of picked-up manure per area (Figure 7-2) and times of last visit (Figure 7-3), we conclude that in experiment one, the longest time manure is on the ground before picking up is between 4 hours and 4 hours and 15 minutes, and the maximum amount in experiment one is 2.75 liters.

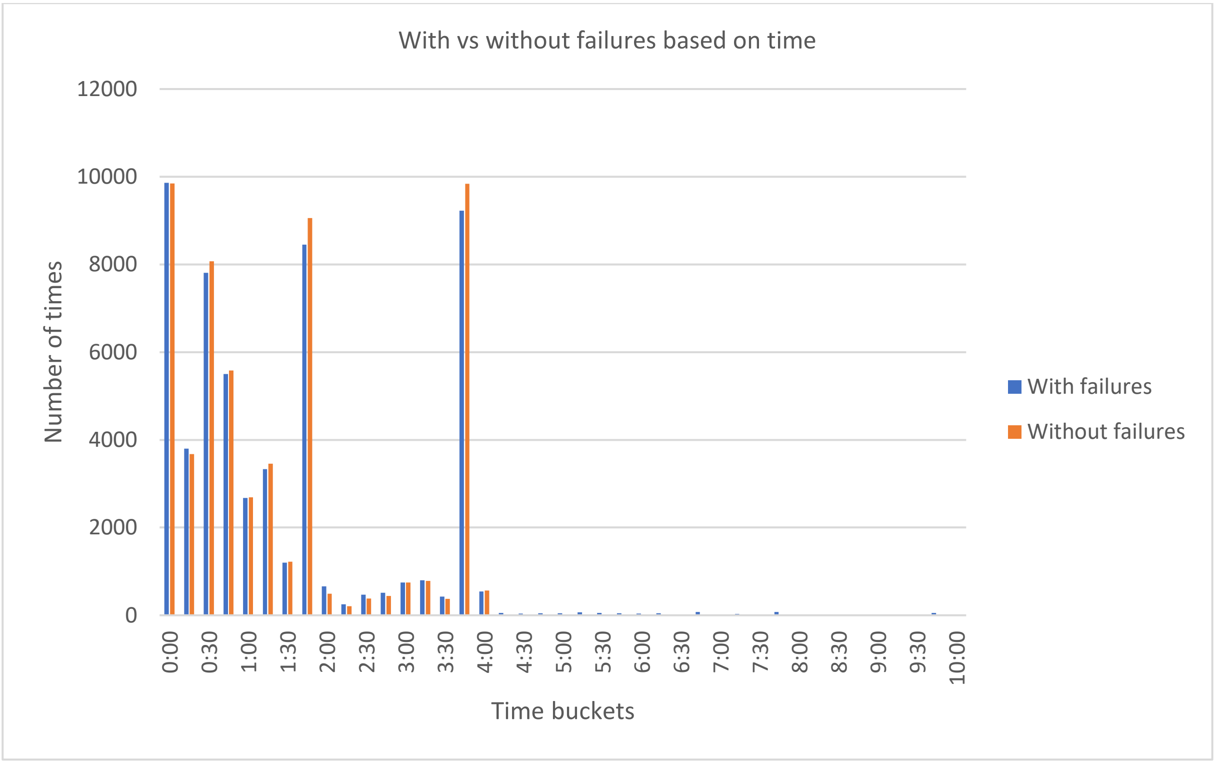


Figure 7-2: Histogram with all times of the last visit per area

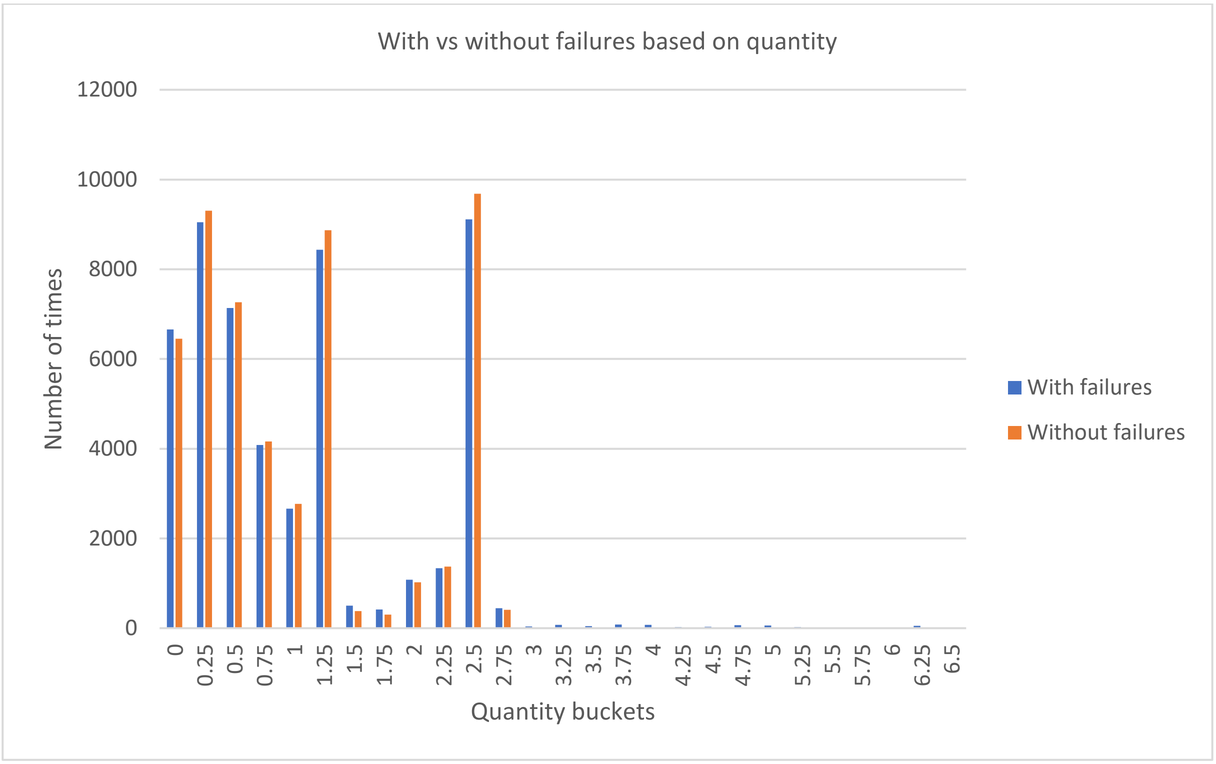


Figure 7-3: Histogram with all quantities of picked-up manure per area

These maximums values are higher in experiment two, up to 10 hours for the longest time and 6.75 liters for the maximum quantity. In experiment two, there were, on average, 746.75 pick-up actions later than 4 hours and 15 minutes, exceeding 592 times the quantity of 2.75 liters on average. How these outliers are spread can be seen in Figure 7-4 and Figure 7-5.

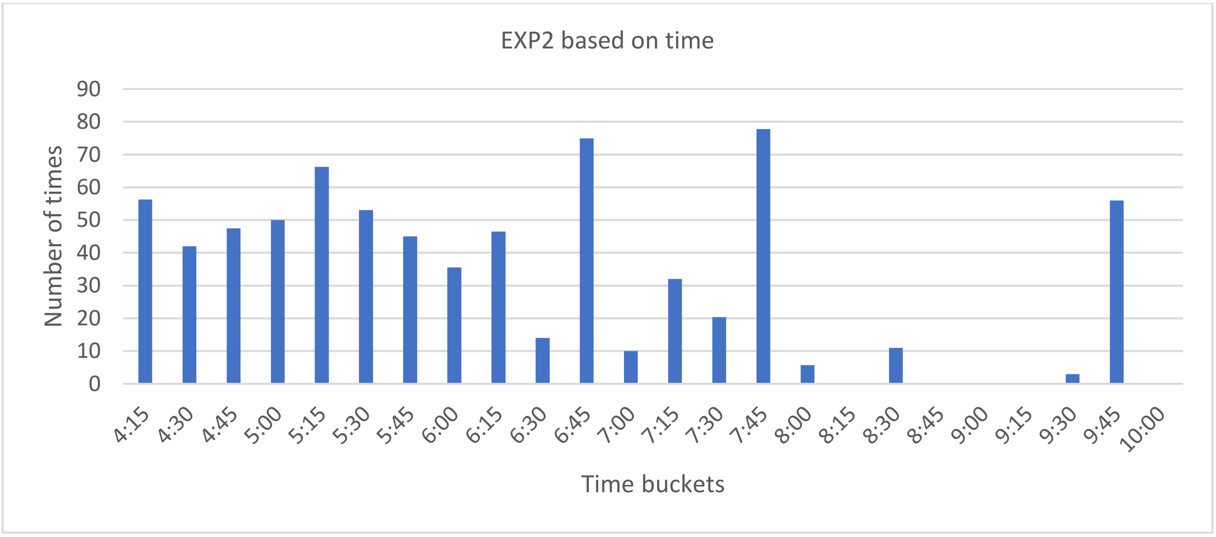


Figure 7-4: Histogram with all times of the last visit per area, focused on the outliers

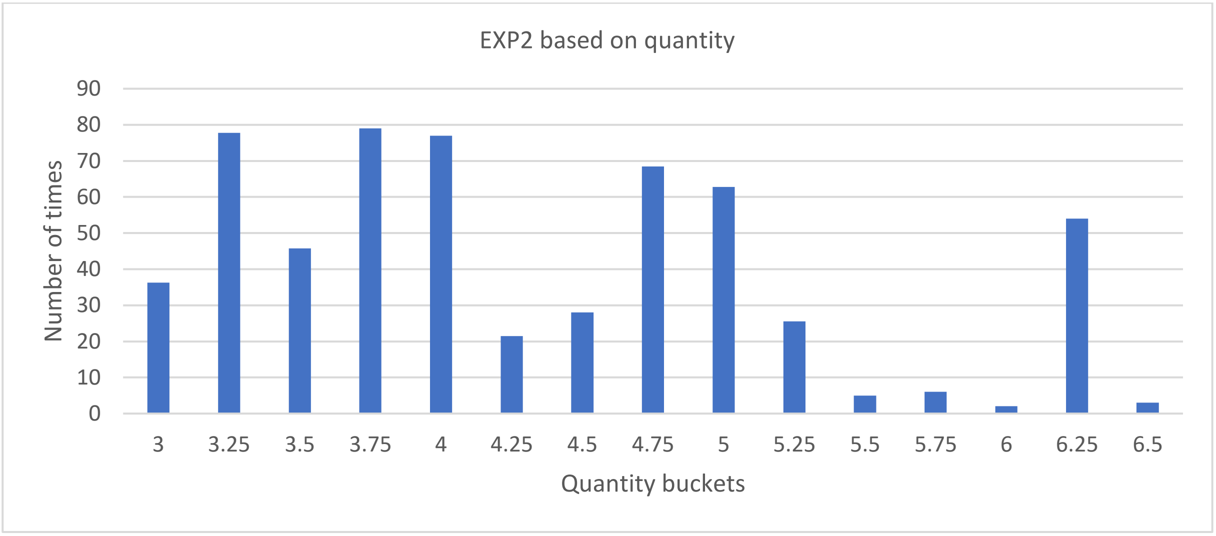


Figure 7-5: Histogram with all quantities of picked-up manure per area, focused on the outliers

7.2. Impact of communication ranges

This section discusses the first part of experiment three, where task handovers and shared facilities are introduced. Of course, we are looking into the impact this has on the average quantity of manure per marker and the impact of the different communication ranges. But in this experiment, the other KPIs are also becoming more critical. This experiment also focuses on the number of times an AGV took over a route and the response time from when a failure occurred until an AGV responds to it by starting a takeover route.

Table 7-1 shows the results of the first part of the third experiment. In this part, all sectors are considered when picking a sector to clean when a failure is detected in the system. From this table, it can be seen that as soon the communication improves (ranges are increasing with every experiment), the number of take-over routes increases, and the average quantity of manure decreases. From one point (experiment 3-8), the communication is at such a level that the system does not improve anymore. This result also suggests that when the communication is at such a level, it does not matter anymore if the MAS is implemented with the hybrid approach of the central approach.

Another important conclusion from these results is that the KPI does not improve compared with experiment two. This result is due to the travel times of AGVs. The travel time becomes more if an AGV has to clean at the other side of the barn, where it usually does not have to clean. This extra travel time takes time, time that cannot be used to clean efficiently. And above that, extra charging time is needed for this traveling. Another reason is that the heuristic does not pick the sector to clean optimally. The smaller sectors are often chosen as most urgent, resulting in only a slight reduction of manure in the barn compared with cleaning a larger sector that is only a bit less urgent. Therefore we are introducing priority sectors in the second part of this experiment (Section 7.3).

Also, an interesting thing that we can make up from this data is that when the communication range with the central system increases, the AGV to AGV communication range has less and less impact on the KPIs. And it is quite logical that when the AGVs have a continuous connection with the central system, AGV to AGV communication does not have any impact anymore. We see also that the different communication ranges have only a minor impact on the average quantity of manure per area but a much more significant impact on the outliers of the number of cleanings after 4:15:00 and cleanings larger than 2.75 liters.

Table 7-1: Results experiment 3a

Experiment	Average quantity per area	With penalty	Average number of failures	Average number of take over routes	# cleanings after 4:15:00	# cleanings larger than 2.75 liters
3-1	0.959	3.452	3.25	5	683.00	648.00
3-2	0.960	3.464	“”	5.625	703.50	668.58
3-3	0.960	3.468	“”	5.375	650.92	618.92
3-4	0.957	3.440	“”	5.5	620.58	585.75
3-5	0.959	3.460	“”	5.625	652.17	617.33
3-6	“”	3.460	“”	5.625	652.17	617.33
3-7	“”	3.458	“”	6	641.42	634.67
3-8	“”	“”	“”	“”	“”	“”
3-9	“”	“”	“”	“”	“”	“”
3-10	“”	“”	“”	“”	“”	“”
3-11	“”	“”	“”	“”	“”	“”
3-12	0.958	3.453	“”	“”	654.33	617.17

7.3. Impact of routing strategy

This section discusses the impact of the three different routing strategies. The first one is the standard arc routing problem (experiment 3-8). This strategy is used to determine the best communication strategy. The results of this strategy can be found in the previous section (Section 7.2). The second strategy is splitting the larger sectors into multiple smaller ones (experiment 3-13). And the third strategy is where specific sectors have priority (experiment 3-14).

Comparing the benchmark with the outcome of experiment 3b, the average quantity of manure per area and the penalty value increases at all three different strategies. This outcome seems to be bad, but when comparing the outliers of the number of cleanings after 4:15:00 and the number of cleanings with a quantity of manure larger than 2.75, we can see that experiment with prioritizing is improving the system. This reduction is advantageous in practice because manure is cleaned quickly, and manure's chance to dry is smaller. For the experiment where the sectors are split up, we conclude that this negatively impacts the system's performance.

Table 7-2: Results experiment 3b

Experiment	Average quantity per area	With penalty	Average number of failures	Average number of take over routes	# cleanings after 4:15:00	# cleanings larger than 2.75 liters
3-8	0.959	3.458	3.25	6	641.42	634.67
3-13	0.965	3.494	“”	11.5	809.500	787.500
3-14	0.974	3.553	“”	11.75	609.917	622.500

7.4. Impact of full autonomy

In this subsection, we discuss the impact of a fully autonomous plannings strategy without pre-programmed routes. The heuristic, which is explained in Section 4.5.1, makes the routes in this system. In this experiment, we made it optional to connect two or more smaller adjacent arcs with each other.

Comparing the results of this experiment (Table 7-3) with the results of experiment 1 and the best option of experiment 3 (experiment 3-4), we see that the average quantity of manure per area and the penalty value in experiment 4-1 to 4-4 is more significant than in experiments 1 and 3. Also, we conclude from this that connecting two or more adjacent arcs is not improving the system.

Table 7-3: Results experiment 4

Experiment	Average	Penalty	Failures	Connected arcs
1	0.921	3.208	no	-
3	0.957	3.440	yes	-
4-1	0.992	3.691	no	no
4-2	1.122	7.527	no	yes
4-3	1.056	4.127	yes	no
4-4	1.380	6.435	yes	yes

Looking per sector at the average quantity of manure per area (Table 7-4), we see many differences compared with experiments 1 and 3. The most important thing is that the average at the dead-end sectors (sector 5-6, 7-8, and 12-13) is significantly higher than the other sectors, which are even lower in some cases.

Table 7-4: Results experiment 4 sectors

Experiment	Failures	1-2	1-3	2-5	3-4	4-7	5-7	5-6	7-8
1	no	1.105	0.870	1.133	0.633	1.128	1.075	0.854	1.187
3	yes	1.156	0.863	1.192	0.657	1.110	1.087	0.914	1.215
4-1	no	0.802	0.485	0.791	0.876	0.992	0.830	0.878	1.320
4-2	no	0.870	0.558	1.597	1.051	1.048	1.553	1.024	1.410
4-3	yes	0.850	0.523	0.840	0.912	1.048	0.882	0.935	1.364
4-4	yes	1.274	0.595	0.852	1.376	1.269	0.890	1.202	1.929
Experiment	Failures	3-9	4-10	9-10	9-11	10-12	11-12	12-13	
1	no	0.791	0.531	0.567	0.481	0.639	0.782	0.868	
3	yes	0.835	0.568	0.637	0.530	0.688	0.817	0.900	
4-1	no	0.857	0.591	0.912	0.803	0.644	1.009	1.278	
4-2	no	0.436	0.556	1.173	0.546	0.573	1.070	1.312	
4-3	yes	0.887	0.642	1.039	0.871	0.714	1.108	1.319	
4-4	yes	0.458	0.632	1.225	0.543	0.694	1.421	2.005	

7.5. Sensitivity analysis

To see how the system reacts and performs to a changing environment, we conduct a sensitivity analysis. The input variables that we are going to change are discussed in Chapter 6.4.

The first factor that we discuss is the increase in the number of cows living in the barn. We have performed two additional experiments where we both times increased the number of cows in the barn to see if the capacity of the AGVs is still sufficient. The results of this can be seen in Table 7-5. In this table's first three experiments, we ran without sector priority, and the last three did we run with sector priority. In this experiment, the most critical KPI is the number of tank overflows. A tank overflow means that the tank is full before arriving at a dumping spot, so the cleaned-up manure exceeds capacity. When increasing the number of cows from 120 to 135, we can see a slight increase in tank overflows, but minimal. Increasing the number of cows again with 15 to a total of 150 cows, we see that the number of tank overflows drastically increases to an average of 67.75 in six days. Looking at the other KPI's, it is pretty logical that these increase a little.

Table 7-5: Results sensitivity analysis increased number of cows

Experiment	# of cows	Normal	With penalty	# tank overflows
5-1	120	0.959	3.456	5.25
5-2	135	1.079	4.104	8.5
5-3	150	1.199	4.800	67.75
5-4	120	0.961	3.485	5.25
5-5	135	1.082	4.140	8.5
5-6	150	1.202	4.844	68.25

In the second part of the sensitivity analysis, the failure durations and intervals are changed. Again, the first three experiments in Table 7-6 we run without sector priority, and the last three did we run with sector priority. Out of this analysis, we see that reducing the average failure length of the failures during daytime (experiment 5-7 and 5-10) by 50% has more impact than reducing the length of the average failure length of the failures during nighttime (experiment 5-8 and 5-11) with 50%. Increasing the number of failures by reducing the average interval between two failures by 50% (experiments 5-9 and 5-12) blows up the system.

Table 7-6: Results from sensitivity analysis failure changes

Experiment	Normal	with penalty	Average number of failures	Average number of take over routes
5-7	0.946	3.383	3.25	9.5
5-8	0.951	3.399	3.25	10.5
5-9	4.069	117.728	8.75	20.75
5-10	0.951	3.419	3.25	8.5
5-11	0.952	3.408	3.25	9.75
5-12	2.938	75.738	9	28

The benefit of prioritizing the sector can also be seen in his experiment. Experiment 5-9 has an average quantity of manure per marker of 4.069 liters, whereas experiment 5-12 limited to an average of 2.938 liters per area.

7.6. Conclusion

From these experiments, we conclude that the failures in the system result in an increase in the average quantity of manure per marker of 3.66% to 0.955 liters. This increase is caused by 3.5 failures over two AGV's measured over six days.

Giving priority to specific sectors in the barn positively affects the results and performs the best out of the three strategies. The average quantity of manure per area increases but more important, there is a reduction of the outliers in the number of cleanings after 4:15:00 and larger than 2.75 liters.

A fully autonomous system performs a little worse overall than the current situation, but significant differences exist between the different sectors. This difference is mainly due to if a sector is a dead-end or not.

The sensitivity analysis gave us information about the influence of capacity of the AGV's. The current capacity of the barn is 120 cows. The sensitivity analysis results conclude that the robot can handle more with only a slightly negative effect on the KPIs. Furthermore, we conclude that reducing the average failure length of the failures during daytime affects the results than reducing the average failure length during nighttime. At last, we conclude that the system blows up when the interval between failures is made shorter by 50%.

8. Conclusion

In this chapter, we give our conclusions of the research project and answer all of our research questions. We first answer the sub-questions, and all these answers together answer our main research question.

1. **What is a self-organizing system, and how can it be designed for agrobotics?**

A self-organizing system is a system where every individual can make decisions on their own concerning a common goal. These systems can be implemented in various cases, from managing stocks and orders to controlling a fleet of AGV's. For the design of such a system, multiple methodologies are available. This research is the so-called Multi-Agent System (MAS) used (Wooldridge & Jennings, 1995). Multiple design methodologies are available to design such a MAS, and the methodology used is the Prometheus methodology (Padgham & Winikoff, 2004). This decision is made because of the company's experience with the methodology and straightforward overview of tables and visualization.

2. **How would a generic self-organizing system look like for agrobotics with simultaneous driving with pickup and delivery?**

By following the Prometheus methodology, we have created a MAS applicable for a wide range of cases of agrobotics. During the design of the MAS, we have kept in mind that robots with the latest technologies should be able to work with it. These technologies include the use of real-time crop quality measuring (e.g., NIR sensors) and accurate location determining and steering systems (e.g., RTK-GPS). Because of that, our MAS can be implemented at robots using driving lanes and crop yield heatmaps, for example.

We have split up the MAS into three levels, high-level, mid-level, and low-level. The high level can be seen as a kind of cloud connection for data storage. In the mid-level are the more intelligent agents living. The agents living at the low level are the agents who are controlling the robot. In total, six agents are designed:

- The Monitoring and Logging Agent (high-level) is responsible for keeping up logged data and monitoring it. Furthermore, it shares essential data when other agents ask and decides when to block which area and/or facility.
- The Forecasting Agent (high-level) is responsible for forecasting the environment and decides which area is the most urgent to clean. This forecast can be based on previous crop yields or fertilization strategies.
- The Cooperative Agent (mid-level) is responsible for making advanced planning choices for job and/or route dispatching.
- The Vehicle Operating Agent (low-level) is responsible for controlling the basic tasks and functionalities of the robot. Also, this agent decides which route to take next.
- The Robot Status Agent (low-level) is responsible for keeping track of battery/fuel levels and the remaining capacities. Based on this information, the agent decides if the robot can start its next job or has to stop.
- The Location Agent (low-level) is responsible for keeping track of the robot's location and avoiding obstacles. If necessary, it can decide to adapt its route to avoid a collision.

It could be that, for some cases, some agents are not applicable because certain functionalities are not part of the system.

Because communication is sometimes difficult in the environment where agrobotics operate, we made three different approaches to implementing the MAS. A central approach where the intelligence is laying on a central point and the basic functionalities on the robot itself. A significant advantage of this system is that it can implement dynamic routing if continuous communication can be guaranteed. The second approach is the hybrid approach, where part of the intelligence lies on a central system and a part of the robot. In this approach, continuous communication is unnecessary, but the performance improves if the communication between the different robots and the central system improves. The advantage of this system is that if the robot cannot communicate with the central system, it can still make some intelligent decisions. The last approach is the decentral approach. There is not a central system in this approach, but all the robot has the full intelligence. Because all robots have full intelligence, they can always make the best decision. A big downside is that robots' jobs might not be aligned when there is no robot to robot communication (e.g., robots are out of range or there are communication problems), and robots can therefore conflict with each other.

3. **How would a simulation model look like for a self-organizing system for agrobotics?**

To simulate the actual situation, we have made a conceptual model for simulation based on the framework of Robinson, (2008). Based on this conceptual model, we made a 3D simulation model in Siemens Plant Simulation. In this model, we are introducing 3 new scenarios:

- Task handovers: in this scenario, an AGV can clean areas that another AGV usually cleans. The task is only handed over when one of the AGVs has a failure, and that failure is known. This scenario has to be implemented at the same time as the following scenario.
- Shared facilities: in this scenario, it becomes possible to share lanes and dump spots. In that case, it becomes possible for AGV's to drive over lanes and dump their manure at the dumping spot, which is usually only used by another AGV.
- Fully autonomous: in this scenario, the AGV has no fixed schedule with preprogrammed routes anymore but only uses a heuristic to make and plan its routes.

These three scenarios have been tested during a simulation study. The simulation has been split up into four experiments.

- The first experiment was equal to the current situation without failures.
- The second experiment was equal to the first experiment, but this time with failures. This experiment has been conducted to gain more insights into the impact of the failures on the system. Furthermore, we use this as a benchmark for the other experiments.
- The third experiment is split into three parts: a common arc routing situation, a situation where priority is given to specific sectors, and a situation where the larger arcs are split up. In this experiment, we did vary the communication ranges of the AGV to AGV and from AGV to the central system to gain more insight into the impact of the different communication ranges. In this experiment, we also introduced the first two scenarios.
- In the fourth experiment, the third scenario is introduced. This scenario also has multiple options. The first is the standard arc routing problem, the second is the option to combine multiple adjacent arcs, and the last option is to give ownership of arcs to an AGV.

4. **How is a self-organizing control system for agrobotics performing?**

The self-organizing control system is, for the most significant part, controlled by two heuristics. The first heuristic is triggered when an AGV comes across another AGV with a failure during a route. In that case, the AGV looks on its own for a new route to its end location, a dumping spot, or the charger. The second heuristic, the optimization heuristic, is triggered when there is a known failure in the environment. This heuristic makes a new route for an AGV covering the most urgent part of the barn to clean. This part of the barn is determined based on the last visit time. This heuristic has three different strategies to chose the arcs. The first one is the standard arc routing. The second one prioritizes arcs, and the third is splitting up the larger arc into smaller ones.

With the first two experiments, the impact of failures in the system is determined, and the benchmark is set for the other experiments. From these two experiments, we conclude that there is on average 0.921 liters of manure per area, and the average penalty value is 3.208 per area in the case without failures. The second experiment, which was equal to the first only with failures, was the average liters of manure per marker 0.955, which is an increase of 3.66%, and the penalty value was 3.422, which is an increase of 6.67%. This increase was all due to 3.5 failures per six days over two AGV's. Comparing the number of cleanings at specific times and the number of cleaned quantities with each other, we conclude that the outliers are after 4:15:00 and more than 2.75 liters. The benchmark for this is 746.75 pick-up actions later than 4 hours and 15 minutes, exceeding 592 times the quantity of 2.75 liters over six days.

The best communication configuration is to have at least 10 meters range for AGV to AGV communication combined with at least 10 meters range for AGV to central system communication, where the AGV's can communicate with the central system in the corners of the barn, at the AMS and the chargers. From this, we can also conclude that a hybrid approach with enough communication possibilities performs just as well as a central approach.

Giving priority to specific sectors in the barn gives the best results considering the minimum number of outliers. On average, there will be more manure in the barn, independent of which heuristic strategy is used, but the maximums are minimized. On average, the number of times an area is cleaned after 4:15:00 is reduced to 641.42 times, and on average, 643.67 times more than 2.75 liters is cleaned.

The fully autonomous scenario performs a little worse overall than the current situation, but significant differences exist between the different sectors. This difference is mainly due to if a sector is a dead-end or not. So here are the maximums minimized.

From the sensitivity analysis, we conclude that 120 cows for two AGV's are not the maximum capacity of the AGV's. The system performs almost as well in a barn with 135 as with 120. Next to that, we conclude that reducing the average duration of failures during daytime has a more positive impact on the KPI's than reducing the average duration of failures during nighttime. At last, can we conclude from the sensitivity analysis that the interval must not get much shorter; otherwise, the system will blow up.

9. Recommendations and further research

In this chapter, we give our recommendations and what could be done for further research. We have split this up into two categories, we start with the theoretical recommendations, and after that, we present the practical recommendations.

Theoretical:

- The simulation model can be expanded in multiple ways. The first one is to model the behavior and visualizations of the cows into the model. If this modeling is done at the detailed level where the daily routines of the cows are modeled precisely, and the moments when and where they relieve their manure and urine, we can build a way more precise heatmap where the barn needs to be cleaned and when.
- The second expansion can be by adding other and different types of autonomous robots into the system. Robots like the feeding robot, AMS, or feed pusher can give much information about when and where it is more crowded in the barn. With this information, the cleaning robots can adapt their schedules to this. For example, the feeding robot gives a message that it starts feeding in 30 minutes. Then it is crowded at the feeding fence. The cleaning robots can adapt their schedule to clean beforehand and clean afterward not to interrupt the cows.
- When the first recommendation is implemented, it is also possible to expand the heuristic with a self-learning part. The heuristic has to keep up information about when and where it picks up how much manure. If enough of this data is logged, the heuristic can make a forecast when and where is laying how much manure. In this way, the cleaning robots can adapt their routes better, and the whole system's performance increases.
- To test the simulation model and the MAS in a different and larger barn where more than two cleaning robots work to see if the system is still performing and in this simulation study and as intended.
- Test the genericity of the MAS. The MAS should also be tested in a simulation study of another case, for example, H2Trac.
- Related to the heuristic, investigate the possibility to apply the heuristic to different cases outside the agriculture sector like robot vacuum cleaners and autonomous lawn movers.

Practical:

- First of all, the MAS should be implemented in the current system and tested in real life.
- The Lely should the Discovery 120 expand with a fill level measuring system. This information is beneficial and needed if the system is made more advanced.

Bibliography

Assad, A. A., & Golden, B. L. (1995). *Arc Routing Methods and Applications*. *8*.

Barkema, H. W., von Keyserlingk, M. A. G., Kastelic, J. P., Lam, T. J. G. M., Luby, C., Roy, J. P., LeBlanc, S. J., Keefe, G. P., & Kelton, D. F. (2015). Invited review: Changes in the dairy industry affecting dairy cattle health and welfare. *Journal of Dairy Science*, *98*(11), 7426–7445. https://doi.org/10.3168/jds.2015-9377

Bartholdi, J. J., Eisenstein, D. D., & Lim, Y. F. (2010). Self-organizing logistics systems. *Annual Reviews in Control*, *34*(1), 111–117. https://doi.org/10.1016/j.arcontrol.2010.02.006

Busoniu, L., & Tamás, L. (2016). *Handling Uncertainty and Networked Structure in Robot Control*. https://doi.org/10.1007/978-3-319-26327-4

Chevalier, A., Copot, C., De Keyser, R., Hernandez, A., & Ionescu, C. (2015). *A Multi Agent System for Precision Agriculture*. 388. https://doi.org/10.1007/978-3-319-26327-4

Essink, B. (2014). *Scenario analysis of a ice and snow control program on the tactical level* (Issue July).

Janani, A., Alboul, L., & Penders, J. (2016). Multi-agent cooperative area coverage: Case study ploughing. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 1397–1398.

Karsten, B., & Tastowe, F. (2020). *Boerderij Gps-test : vooral verschil in bedieningsgemak*. https://www.boerderij.nl/Mechanisatie/Achtergrond/2020/7/Gps-test-vooral-verschil-in-bedieningsgemak-607553E/

Law, A. M. (2015). *Simulation Modeling and Analysis* (fifth).

Padgham, L., & Winikoff, M. (2004). Developing Intelligent Agent Systems. In *Developing Intelligent Agent Systems*. https://doi.org/10.1002/0470861223

Robinson, S. (2008). Conceptual modelling for simulation Part II: A framework for conceptual modelling. *Journal of the Operational Research Society*, *59*(3), 291–304. https://doi.org/10.1057/palgrave.jors.2602369

Sanaa, M., Poutrel, B., Menard, J. L., & Serieys, F. (1993). Risk Factors Associated with Contamination of Raw Milk by Listeria monocytogenes in Dairy Farms. *Journal of Dairy Science*, *76*(10), 2891–2898. https://doi.org/10.3168/jds.S0022-0302(93)77628-6

Shehory, O., & Sturm, A. (2001). Evaluation of modeling techniques for agent-based systems. *Proceedings of the International Conference on Autonomous Agents*, 624–631. https://doi.org/10.1145/375735.376473

Sipahioglu, A., Kirlik, G., Parlaktuna, O., & Yazici, A. (2010). Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach. *Robotics and Autonomous Systems*, *58*(5), 529–538. https://doi.org/10.1016/j.robot.2010.01.005

Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, *22*(3), 329–337. https://doi.org/10.1016/0377-2217(85)90252-8

Vaddella, V. K., Ndegwa, P. M., Joo, H. S., & Ullman, J. L. (2010). Impact of Separating Dairy Cattle Excretions on Ammonia Emissions. *Journal of Environmental Quality*, *39*(5), 1807–1812. https://doi.org/10.2134/jeq2009.0266

Villettaz Robichaud, M., de Passillé, A. M., Pellerin, D., & Rushen, J. (2011). When and where do dairy cows defecate and urinate? *Journal of Dairy Science*, *94*(10), 4889–4896. https://doi.org/10.3168/jds.2010-4028

Wagner-Storch, A. M., & Palmer, R. W. (2003). Feeding behavior, milking behavior, and milk yields of cows milked in a parlor versus an automatic milking system. *Journal of Dairy Science*, *86*(4), 1494–1502. https://doi.org/10.3168/jds.S0022-0302(03)73735-7

Winston, W. L. (2004). *Operations research*. https://doi.org/10.1016/S0076-5392(08)62705-8

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *10*(2), 115–152. https://doi.org/10.1017/S0269888900008122

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, *12*(3), 317–370. https://doi.org/10.1145/958961.958963

Appendix

Appendix A. Functionalities

Down here are all the functionalities described with their goal, action, trigger, which information is used and produced.

Determining remaining capacity	**Lely case**
Description	This functionality determines the remaining capacity of the robot
Goal	To get to know the remaining capacity of the robot
Actions	Continue working or drop off goods
Triggers	Continuous measurement
Information used	Information from the fill level sensor
Information produced	Remaining capacity

Determining fuel/battery level	**Lely case**
Description	This functionality determines the remaining fuel/battery level of the robot
Goal	To get to know for how long the robot can continue working
Actions	Continue working or go to recharge/refuel place
Triggers	Continuous measurement
Information used	Battery sensor
Information produced	The remaining working time of the robot

Obstacle detection	**Lely case**
Description	This functionality makes sure that the robot detects obstacles on its path and does not drive into them.
Goal	Don’t hit obstacles
Actions	Detecting obstacle and wait till it is gone away or drive around it.
Triggers	When the sensors are detecting something
Information used	Information from the detection sensor
Information produced	-

Determining location	**Lely case**
Description	This functionality localizes the robot in the environment where it is currently working.
Goal	To get to know the location of the robot
Actions	Continue working or change job-based on the location
Triggers	Continuous measurement
Information used	Information from multiple sensors and usage of the SLAM algorithm
Information produced	The location of the robot

Pause/resume	**Lely case**
Description	This functionality pauses the current job of the robot for a particular time and resume its job afterward
Goal	To pause the robot, so there is no conflict with other robots
Actions	Pause and resume job
Triggers	Other robot is close by or is on the same destination
Information used	Location, remaining capacity, remaining working time of the robot
Information produced	-

Determine next route	**Lely case**
Description	This functionality determines the next route the robot has to drive. The destination can be the start point of a job, dumping spot, or the route it has to drive during collecting/receiving.
Goal	To create a close to the optimal route to the destination (with performing a job).
Actions	Picking the next route to drive
Triggers	End/start of daily routing, end of the route, or finished charging
Information used	Technician configuration (set-up), Continue working or change job, Remaining capacity, The remaining working time of the robot, Location
Information produced	-

Retrieve/collect goods	**Lely case**
Description	This functionality makes sure that the robot is Retrieving/collecting goods at the correct place at the correct time while driving.
Goal	Collecting/receiving goods
Actions	Drive to the correct place in the barn and clean up manure while driving
Triggers	End/start of daily routing, end of the route, or finished charging
Information used	Technician configuration (set-up), Continue working or change job, Remaining capacity, The remaining working time of the robot, Location
Information produced	-

Dump goods	**Lely case**
Description	Dump goods so it can collect goods again.
Goal	Expanding remaining capacity
Actions	Go to dumping spot and lose all goods
Triggers	No/low remaining capacity
Information used	Technician configuration (set-up), Continue working or change job, Remaining capacity, The remaining working time of the robot, Location
Information produced	-

Go to start/end location	**Lely case**
Description	When the route/job is finished, the robot has to go to the start/end location
Goal	To let the robot go to the start point
Actions	Drive to the start/endpoint of all the routes and charge the vehicle
Triggers	When dumping and the route is finished
Information used	Location, remaining capacity, remaining working time of the robot
Information produced	-

Increase/decrease of active robots in the system	**Lely case**
Description	This functionality makes an inactive robot active or takes an active robot out of the system by making it inactive.
Goal	To have the number of robots in the system equals the need for the system at that moment.
Actions	Controlling the number of active robots in the system
Triggers	Needed capacity changed
Information used	Lack/surplus of capacity
Information produced	-

Route adaption	**Lely case**
Description	This functionality changes the routes of the robots when it is needed and possible
Goal	To let the robots work and drive close to optimal routes
Actions	Look for routes with better performance than the current routes
Triggers	Changes in vehicle status or form high-level
Information used	Vehicle status (all vehicles), Configuration goal, Manual locks, Technician configuration (set-up), location
Information produced	Continue working or change job

Cooperative despatching	**Lely case**
Description	This functionality is reallocating the robots when some significant changes happened in the system
Goal	Reallocate the robots in such a way that the system is performing close to optimal
Actions	Determining needed and current capacity
Triggers	When a robot is connected with the network and something is changed in the environment, or something is changed in the settings from above
Information used	Forecast of the environment
Information produced	Lack/surplus of capacity, continue working or change job

Forecasting	**Lely case**
Description	This functionality makes a forecast of how the environment looks like in the following time period
Goal	To make a good prediction based on historical data how the environment looks like
Actions	Make forecast
Triggers	When new input arrives
Information used	Vehicle status (all vehicles), manual locks
Information produced	Forecast of environment

Monitoring and logging	**Lely case**
Description	This functionality monitors and logs the status and performance of the robots
Goal	To get a good overview of the performance of the robots
Actions	Gather the status of the robots
Triggers	Information is shared when the robot is connected to the server.
Information used	Vehicle status of all vehicles
Information produced	-

Appendix B. Agent descriptors

Down here are all the agent descriptors are shown with all their characteristics.

Name	**Vehicle operating agent**
Description	This agent is responsible for the primary tasks of the robot
Cardinality	One for every robot
Lifetime	When route and/or job is finished, when the capacity is reached, or when the battery is almost empty
Initialization	When the robot wakes up, it activates the functionalities: determine next job and determine next route
Demise	When the robot is inactive
Functionalities included	Pause/resume, Determine next route, Retrieve/collect goods, dump goods, go to start/end location (charging point)
Uses data	Remaining capacity, The remaining working time of the robot, Continue working or change job and Technician configuration (set-up), Location
Produces data	-
Goals	To keep the robot working on the essential tasks
Percepts responded to	Vehicle stopped, route finished, connection lost, charging station occupied, delivery point occupied
Actions	Stop driving, start driving, request route, open tank close tank
Protocols and Interactions	Cooperative dispatching

Name	**Vehicle status agent**
Description	This agent keeps track of the status of the fuel/battery level and the remaining capacity of the robot
Cardinality	One for every robot
Lifetime	When the robot is active
Initialization	Determines the fuel/battery level and the remaining capacity of the robot when it wakes up
Demise	Agent shuts down when the robot becomes inactive
Functionalities included	Determining remaining capacity and Determining fuel/battery level
Uses data	Battery sensor, Information from the fill level sensor
Produces data	Remaining capacity, the remaining working time of the robot, and the vehicle status
Goals	To make sure the robot does not run out of power and do not reaches the maximum capacity during a job
Percepts responded to	Manure tank full, manure tank empty, battery level low, battery full, water tank empty, water tank full,
Actions	Request delivery spot
Protocols and Interactions	-

Name	**Location agent**
Description	This agent tracks the location of the robot and checks if there are no obstacles in its path
Cardinality	One for every robot
Lifetime	When the robot is active
Initialization	Determines the location when the robot wakes up
Demise	Agent shuts down when the robot becomes inactive
Functionalities included	Obstacle detection, Determining location
Uses data	Location sensors
Produces data	Location
Goals	To keep track of the location of the robot and that the robot does not ride into an obstacle
Percepts responded to	Vehicle stuck, location unknown
Actions	Stop driving, start driving, reallocate task
Protocols and Interactions	-

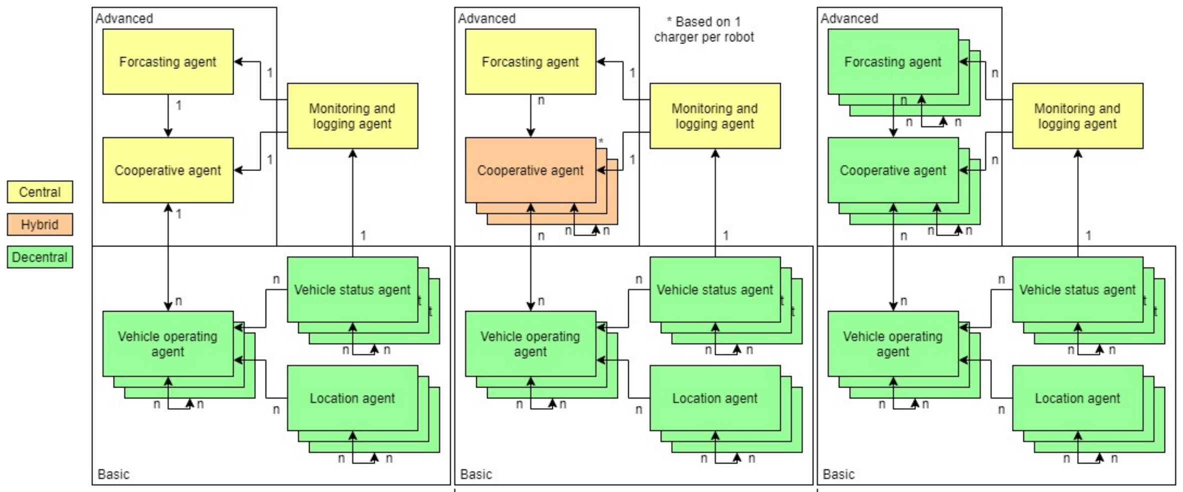
Name	**Cooperative agent**
Description	This agent is the agent who allocates the robot to jobs and areas
Cardinality	Depended on the approach
Lifetime	Till the robots are allocated again
Initialization	During the initialization, the agent has to gather all needed information to determine the new strategy
Demise	Can close when robots are allocated
Functionalities included	Route adaption, Cooperative dispatching, Increase/decrease number of active robots in the system
Uses data	Vehicle status, Forecast of environment, Technician configuration (set-up), Manual locks, Configuration goal, Location
Produces data	Continue working or change job, Lack/surplus of capacity
Goals	To allocate the robots in such a way that the system performs close to optimal concerning the configuration goal
Percepts responded to	-
Actions	Reallocate task
Protocols and Interactions	Cooperative dispatching

Name	**Forecasting agent**
Description	This agent makes a forecast of the environment
Cardinality	Depended on the approach
Lifetime	Is woken up when sufficient data is gathered and dies when a forecast is made
Initialization	During the initialization, the agent has to retrieve all the vehicle status
Demise	Can close when a forecast is made
Functionalities included	Forecasting
Uses data	Vehicle status, manual locks
Produces data	Forecast of environment
Goals	To get a forecast of the environment
Percepts responded to	-
Actions	Reallocate task
Protocols and Interactions	-

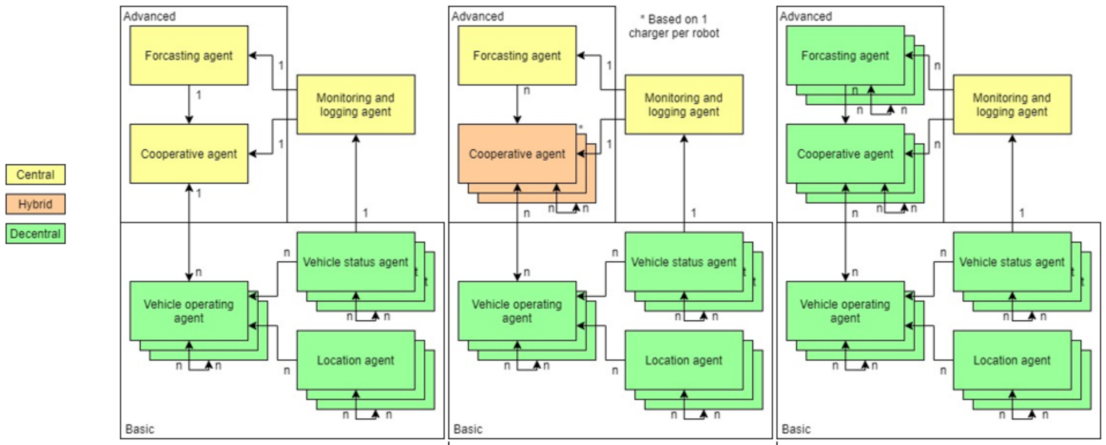
Name	**Monitoring and logging agent**
Description	Monitors and logs the performance of the robots
Cardinality	One for the whole system
Lifetime	It is woken up when the robot connects to the server and dies when synchronization is done
Initialization	When initializing, the agent has to identify the robot and retrieve all data of the robot
Demise	Can close when synchronization is done
Functionalities included	Monitoring and logging
Uses data	Vehicle status
Produces data	-
Goals	To get to know the performance of each robot and the system
Percepts responded to	-
Actions	-
Protocols and Interactions	-

Appendix C. Central vs. Decentral vs. Hybrid

Here are all the characteristics shown of the different communication approaches.



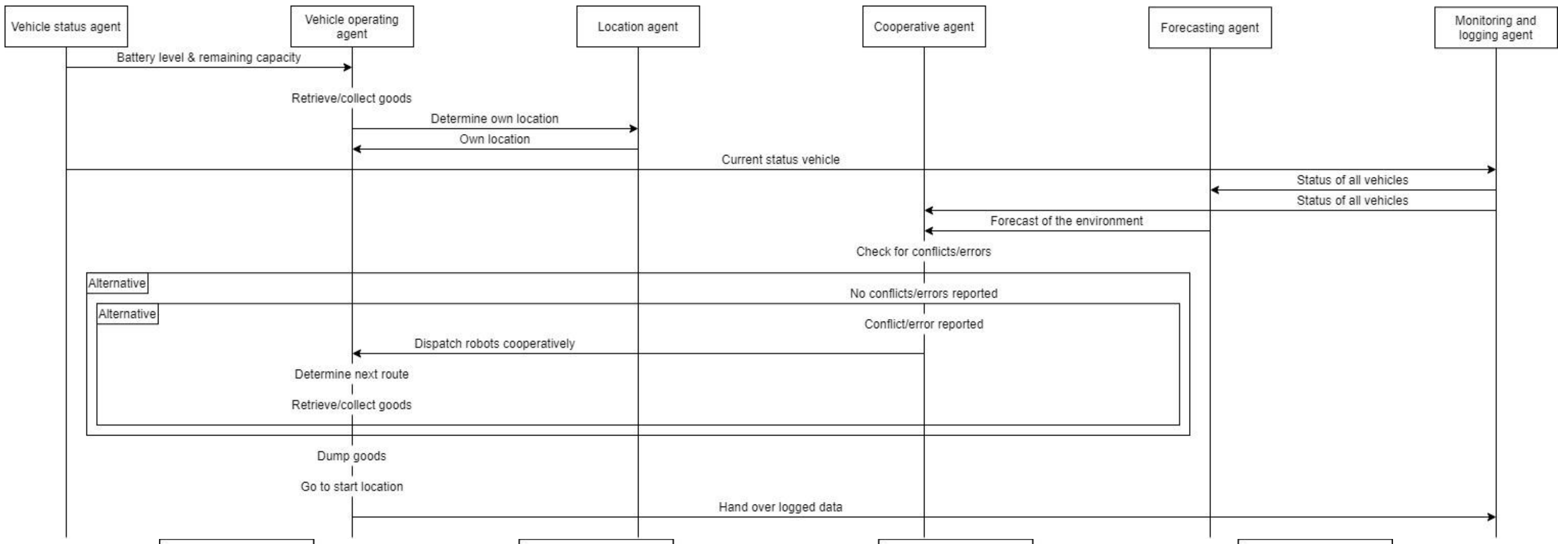
Design	Central	Hybrid	Decentral
Basic functionalities	Static routing and neighbourhood communication for conflict avoidance in shared places	Static routing and neighbourhood communication for conflict avoidance in shared places	Static routing and neighbourhood communication for conflict avoidance in shared places
Advanced functionalities	Dynamic routing and optimization of routes for all robots due to continuously connection between the central system and all robots	Static routing with the possibility to choose from an extra set of route which cover areas form other robots, the possibility to gain information about routes of other robots while charging and the option to go look for a robot when it did came back on time at the charging spot	Static routing with the possibility to choose from an extra set of route which cover areas form other robots

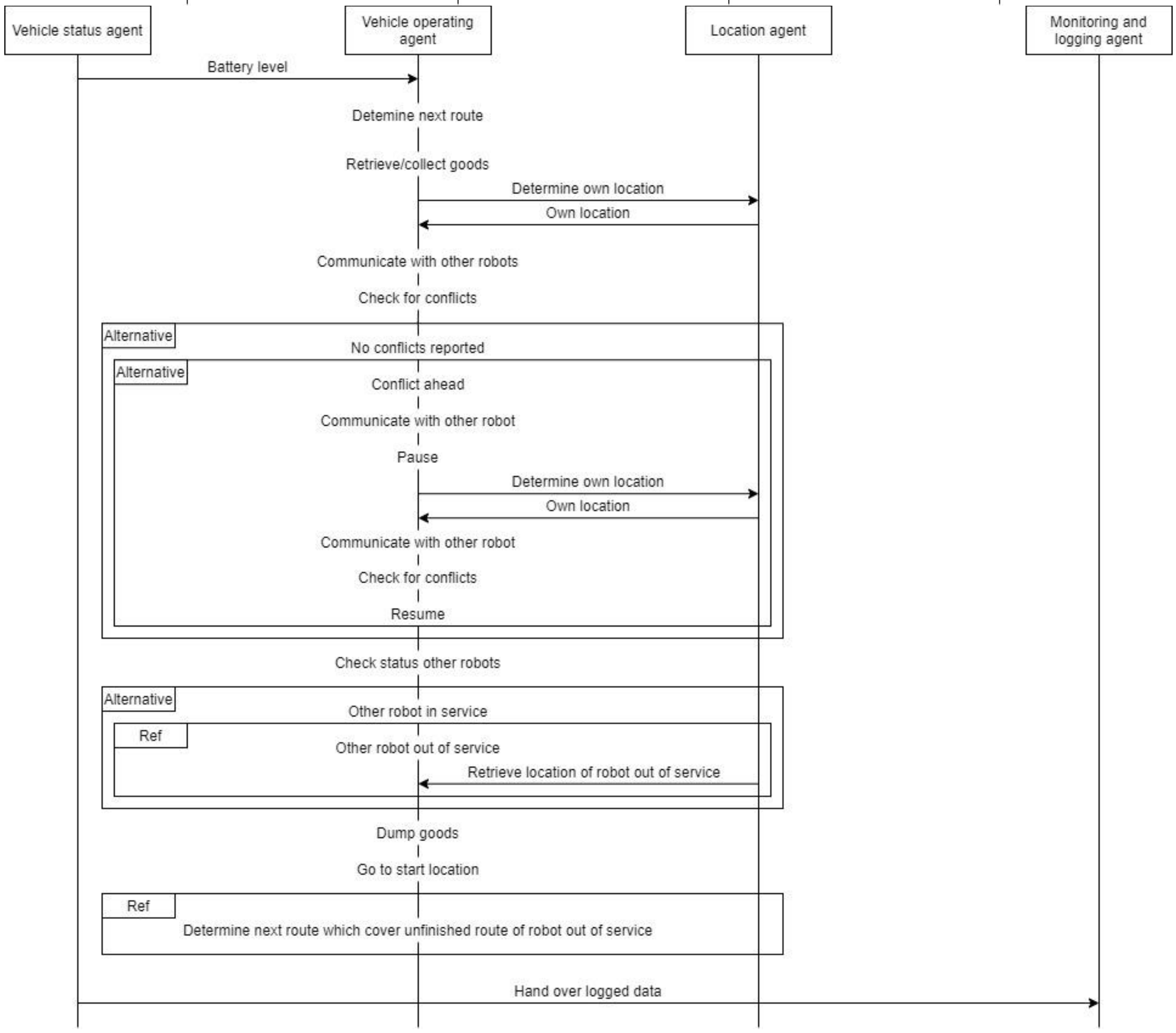


	Design	Central	Hybrid	Decentral
	Number of links exterr	9n	12n	15n
	Number of links intern	5+3n	3+5n	1+7n
	Link density	7/15	7/15	7/15
	SPOF	Low chance in the basic functionalities, High chance in the advanced functionalities	Low chance in the basic functionalities, Medium to high chance in the advanced functionalities	Low chance in the basic functionalities, Medium to low chance in the advanced functionalities
Performance	Number of route adaptions	High	Medium	Medium to low
	Number of conflicts	Low	Medium to low	Medium
	Dynamic route adaptions	Possibility to adapt dynamically	-	-
Safe	Number of conflicts	Low	Medium to low	Medium
	Number of route adaptions to avoid conflicts	High	Medium	None
Secure	Number of data transitions	High	Medium	Low
	Number of data transitions between different levels	High	Medium	Low

Appendix D. Agent interaction diagrams

Here are the agent interaction diagrams are shown of the central and decentral approaches.





Appendix E. Messages Descriptors

Down here are all the message descriptors described with their characteristics.

Name	**Vehicle status**
Description	The status of the vehicles (battery level, remaining capacity)
From agent	Vehicle status agent, monitoring, and logging agent
To agent	Monitoring and logging agent, forecasting agent, cooperative agent
Purpose	To share the current vehicle status for logging, monitoring, forecasting, and optimize the system
Information carried	Battery level, remaining capacity

Name	**Location of vehicle**
Description	Sends information about the current location of the robot
From agent	Location agent
To agent	Vehicle operating agent
Purpose	To share the current location of the robot, which is needed for determining the next route/job
Information carried	The location

Name	**The remaining working time of the robot**
Description	This message gives the information about how long a robot can perform his main task concerning the working time
From agent	Vehicle status agent
To agent	Vehicle operating agent
Purpose	To share the time the robot can perform his main task
Information carried	The time the robot can perform its main task

Name	**The remaining capacity of the robot**
Description	This message gives information about how long a robot can perform its main task concerning the capacity
From agent	Vehicle status agent
To agent	Vehicle operating agent
Purpose	To share the time the robot can perform his main task
Information carried	The time the robot can perform its main task

Name	**Information of other robots gathered during route**
Description	This message contains the location and data of the other robots if they are stuck or in error to the charger.
From agent	Vehicle operating agent
To agent	Cooperative agent
Purpose	To share information so other robots can adapt their routes to improve the overall performance of the system.
Information carried	Status information of other robots

Name	**Information of other robots gathered during charging**
Description	This message contains the location and data of the other robots if they are stuck or in error to the charger.
From agent	Cooperative agent
To agent	Vehicle operating agent
Purpose	To receive information about other robot status and adapt their routes on that information to improve the overall performance of the system
Information carried	Status information of other robots

Appendix F. Protocols

Down here are all the protocols described with their characteristics.

Name	**Cooperative dispatching protocol**
Description	This protocol gives the command to robots if they have to change their current working routine/job
Scenarios	Start vehicle, stop vehicle, start task, stop task, Schedule, coordinate
Agents	Cooperative agent, vehicle operating agent
Messages	Vehicle status, location of the vehicle
Notes	It depends on the type of system (e.g., central, decentral, or hybrid) in which scenarios, agents, and messages are used.

Name	**Update status**
Description	This protocol checks the status of the other robots, and the vehicle operating agent determines the next route of the robot based on this status
Scenarios	Start vehicle, start task, schedule, coordinate
Agents	Vehicle operating agent, Location agent, vehicle status agent
Messages	Vehicle status, location of the vehicle
Notes	

Name	**Neighborhood checking**
Description	This protocol checks if there are other robots in the neighborhood, which can result in a conflict.
Scenarios	Start vehicle, stop vehicle, start task, stop task, schedule, coordinate
Agents	Location agent, vehicle operating agent, vehicle status agent, cooperative agent
Messages	Location of vehicle, The remaining working time of the robot, The remaining capacity of the robot
Notes	It depends on the type of system (e.g., central, decentral, or hybrid) in which scenarios, agents, and messages are used.

Name	**Location determining protocol**
Description	This protocol determines the location of the robot with a certain accuracy.
Scenarios	-
Agents	Location agent
Messages	Location of vehicle
Notes	-

Name	**Forecasting protocol**
Description	This protocol makes a forecast of the environment of the capacity needed and where to clean first.
Scenarios	-
Agents	Forecasting agent
Messages	Vehicle status
Notes	It depends on the type of system (e.g., central, decentral, or hybrid) if this protocol is used.

Appendix G. Percepts

Down here are all the percepts described with their characteristics.

Name	**Manure tank full**
Description	Occurs when the manure tank is full of the robot
Information carried	That the manure tank of the robot has reached its capacity and the robot cannot pick up any manure anymore
Knowledge updated	One of the active robots cannot pick up any manure anymore temporarily.
Source	Information comes from the manure level sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	After a specific time of working

Name	**Manure tank empty**
Description	Occurs when the manure tank is empty of the robot
Information carried	That the manure tank of the robot is empty again
Knowledge updated	One of the active robots is ready to pick up manure again
Source	Information comes from the manure level sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	After finishing dumping manure

Name	**Battery level low**
Description	Occurs when the battery is almost empty, and the robot just can reach the charging point
Information carried	That the battery is almost empty
Knowledge updated	That the robot has to drive to a charging point to charge
Source	Information comes from the battery sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	After a specific time of working without charging

Name	**Battery full**
Description	Occurs when the battery is done charging
Information carried	That the battery is full again
Knowledge updated	That the robot can be ready to operate again
Source	Information comes from the battery sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	Whenever charging is completed

Name	**Vehicle stopped**
Description	Occurs when the robot is stopped driving, by purpose or not
Information carried	That the robot is not driving anymore
Knowledge updated	One of the active robots cannot drive anymore because of a problem/obstacle. Letting the vehicle stop by purpose is successful
Source	Vehicle operating agent
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle operating agent
Expected frequency	Occasional with medium frequency

Name	**Vehicle stuck**
Description	Occurs when the robot wants and tries to move but is not moving
Information carried	That the robot is not capable of moving anymore
Knowledge updated	The system has one active robot less temporary
Source	Information comes from the driving motor and location determination.
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents and signals to the user.
Agents responding	Location agent
Expected frequency	Occasional with low frequency

Name	**Water tank empty**
Description	Occurs when the water tank is empty of the robot
Information carried	That the water tank of the robot is empty, and the robot cannot spray water anymore.
Knowledge updated	One of the active robots cannot clean properly anymore
Source	Information comes from the water level sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	After a specific time of working

Name	**Water tank full**
Description	Occurs when the water tank is full again
Information carried	That the water tank is full again and the cleaning performance is improved again
Knowledge updated	The robot can clean properly again
Source	Information comes from the water level sensor
Processing	The signal comes in at the robot monitor agent and passes this through to the other agents.
Agents responding	Vehicle status agent
Expected frequency	After finishing filling the water tank

Name	**Route finished**
Description	Occurs when the programmed route is finished
Information carried	That the route is finished and the system can determine the next one
Knowledge updated	Robot can start a new route
Source	Charging station, robot, and location of the robot
Processing	Signal comes in at the charger and robot at the same time. The battery charge and the robot determine its next route at the end of the charging cycle
Agents responding	Vehicle operating agent
Expected frequency	At the end of every route

Name	**Connection lost**
Description	Occurs when the robot has lost its connection towards the system and/or robots
Information carried	Could not connect to robot and/or central system
Knowledge updated	Cannot share data with the complete system
Source	
Processing	Try to reconnect; otherwise, adapt the routes of robots so that possible conflicts are avoided
Agents responding	Vehicle operating agent
Expected frequency	Occasional with medium frequency

Name	**Charging station occupied**
Description	Occurs when the robot had planned to charge, but the station is occupied
Information carried	That the robot cannot charge
Knowledge updated	The line for the charging station
Source	The location agent notice that the station is already occupied
Processing	The second robot has to pause and resume when the other robot is done charging
Agents responding	Vehicle operating agent
Expected frequency	Occasional with low frequency

Name	**Delivery point occupied**
Description	Occurs when the robot had planned to dump its goods, but the dumping spot is occupied
Information carried	That the robot cannot dump goods
Knowledge updated	Queue for the dumping station
Source	The location agent notice that the delivery point is already occupied
Processing	The second robot has to pause and resume when the other robot is done dumping
Agents responding	Vehicle operating agent
Expected frequency	Occasional with very low frequency

Name	**Location unknown**
Description	Occurs when the robot does not know his location anymore
Information carried	-
Knowledge updated	Current location unknown
Source	Location agent
Processing	Try to reconnect
Agents responding	Location agent
Expected frequency	Occasional with medium frequency

Appendix H. Actions

Down here are all the actions described with their characteristics.

Name	**Request delivery spot**
Description	With this action, a robot can request a delivery spot to deliver its goods. The result is that the robot gets a spot assigned where it can go to deliver its goods.
Parameters	The location of the robot and the other robot. If the delivery spots are taken or not.
Temporality	Instantaneous
Failure detection	Yes, could not assign a delivery spot
Partial change	The robot has to wait for a while and take another attempt
Side effects	None

Name	**Stop driving**
Description	With this action, the robot has to stop driving
Parameters	Location, speed, activity
Temporality	Instantaneous
Failure detection	Yes, the robot is still moving
Partial change	Shut down robot completely
Side effects	The robot cannot continue working on the current job

Name	**Start driving**
Description	With this action, the robot starts to drive.
Parameters	Location, speed, activity, route, job
Temporality	Continuous from receiving action until receiving action stop driving
Failure detection	Yes, the robot is not moving
Partial change	There are no results. To get results, the robot has to drive
Side effects	The robot cannot continue working on the current job

Name	**Reallocate task**
Description	With this action, the robot can change from doing one task to doing another task.
Parameters	Location, activity, route, job, status robots, forecast
Temporality	Till the new task is done or when the robot reallocates the task
Failure detection	Yes, when the robot is not moving or capacity levels are not changing
Partial change	This can be different from not performing the task good enough to not driving at all
Side effects	This is depending on what the new task is. It can be that only the battery level drops, but it is also possible that the remaining capacity is dropping over time.

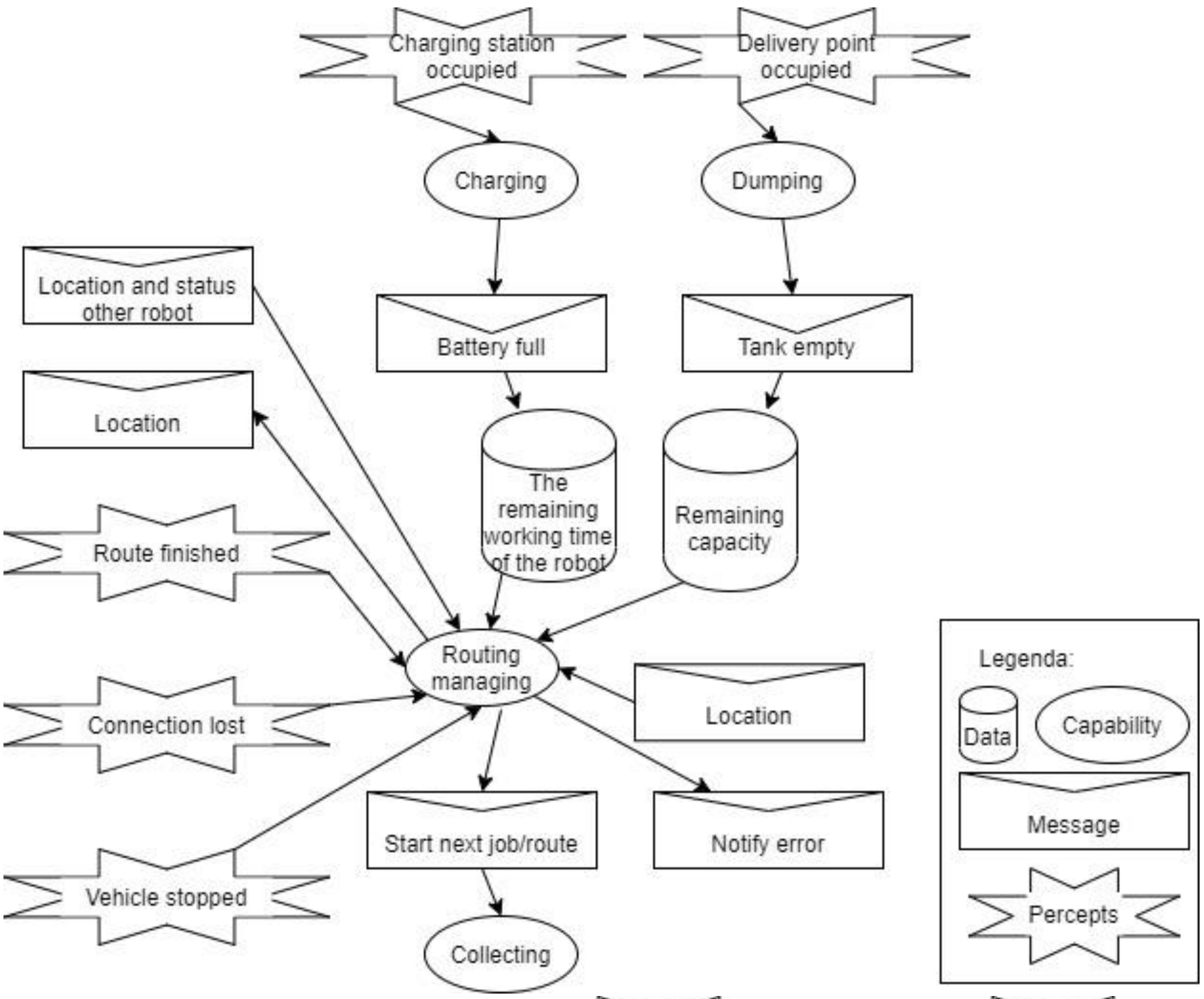
Name	**Request route**
Description	With this action, the robot gets a new route to drive
Parameters	Location, the status of the robots
Temporality	Till the new route is finished
Failure detection	Yes, when the robot is not starting to drive
Partial change	The robot is not driving
Side effects	When driving, the battery level drops

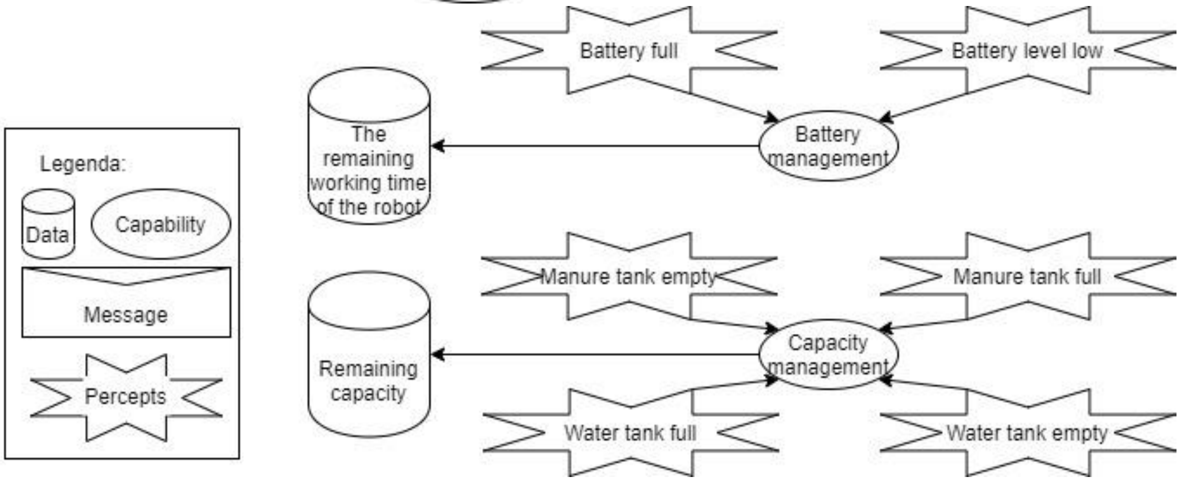
Name	**Open tank**
Description	This action opens the tank of the Lely discovery
Parameters	Location
Temporality	Instantaneous
Failure detection	If loaded, the fill level does not drop
Partial change	If the action does not succeed, then the vehicle has an error
Side effects	The remaining capacity increase till maximum

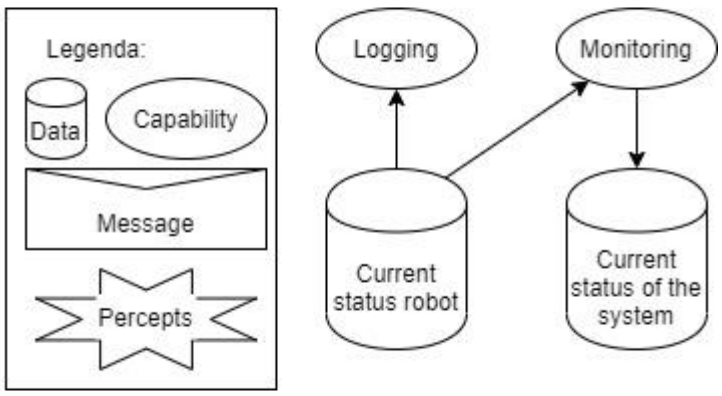
Name	**Close tank**
Description	This action closes the tank of the Lely discovery
Parameters	Location
Temporality	Instantaneous
Failure detection	No
Partial change	If the action does not succeed, then the vehicle has an error
Side effects	None

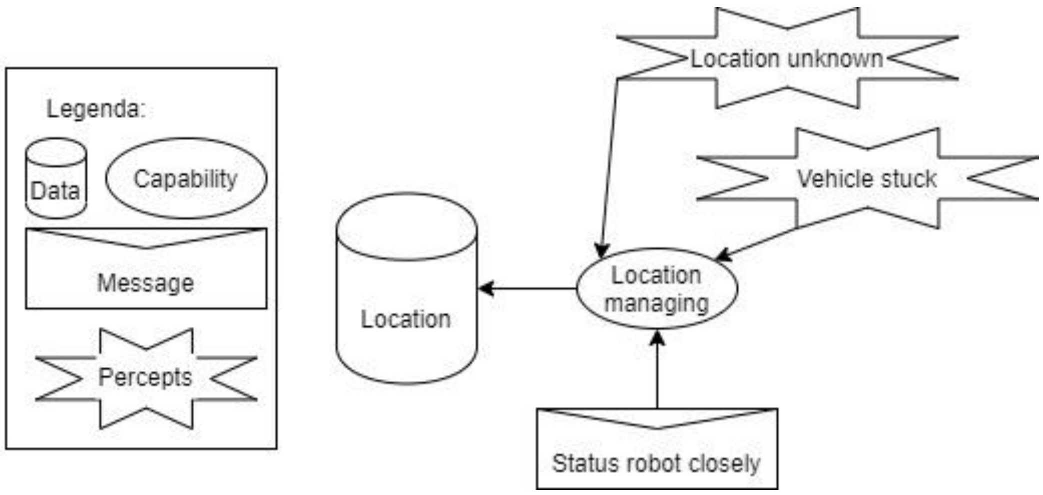
Appendix I. Agent overview diagrams

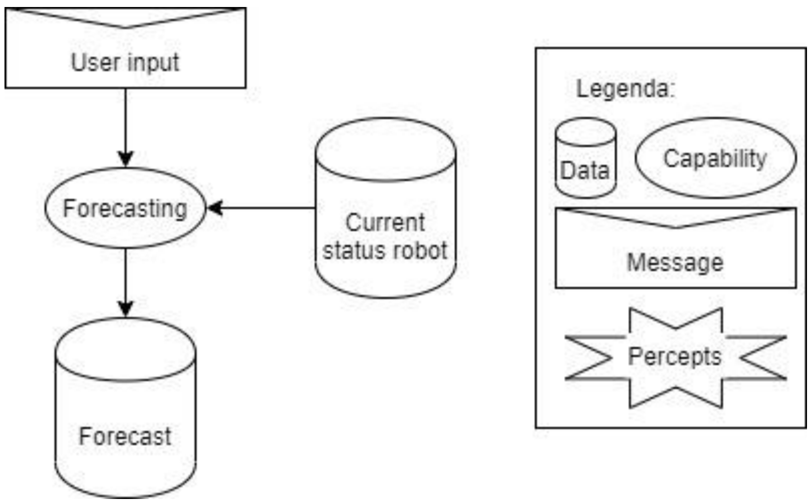
Down here are all the agents visualized in agent overview diagrams.

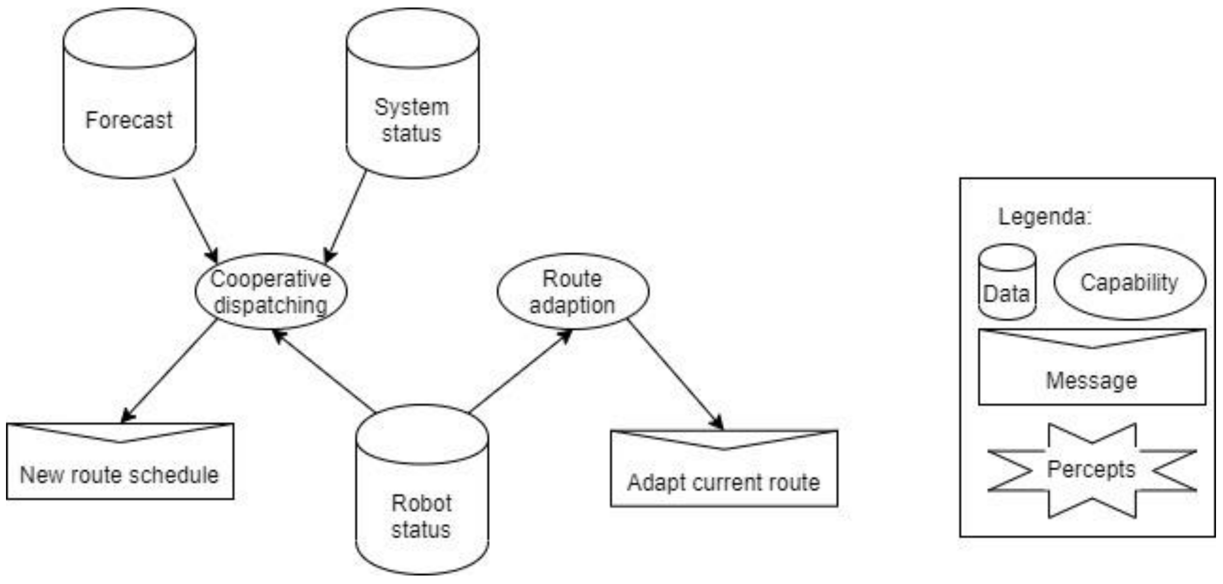





Legenda:
Data
Capability
Message
Percepts
Logging
Monitoring
Current status robot
Current status of the system


Legenda:
Data
Capability
Message
Percepts
Location unknown
Vehicle stuck
Location
Location managing
Status robot closely


User input
Forecasting
Current status robot
Forecast
Legenda:
Data
Capability
Message
Percepts


Forecast
System status
Cooperative dispatching
Route adaption
New route schedule
Robot status
Adapt current route
Legenda:
Data
Capability
Message
Percepts

Appendix J. Capability descriptors

Down here are all the capabilities descriptors described with their characteristics.

Name	**Routing managing**
Description	This capability selects the next route for the robot based on the information it got
Goals	selecting the next route for the robot
Processes	Determining the next route
Protocols	Cooperative dispatching
Outgoing messages	Start route x
Incoming messages	Battery full, Tank empty
Internal messages	Which sectors are going to be combined to create in the next route, and which sectors are going to be cleaned in total, and which is a pass-through sector
Percepts	Route finished, connection lost, Vehicle stopped
Actions	Select the next route
Included capabilities	-
Data used: imported	Battery level, status other robots
Data internal	-
Included capabilities	-
Included plans	-
Notes	-

Name	**Route adaption**
Description	This capability decides if a route needs to be adapted if two robots are getting close to each other
Goals	Avoid collisions
Processes	Determine if the route needs to be adapted
Protocols	-
Outgoing messages	Adapt current route
Incoming messages	-
Internal messages	Danger, No danger
Percepts	-
Actions	-
Included capabilities	Check if robot might have a collision soon, Stop the current route and set a new destination
Data used: imported	Robot status
Data internal	-
Included capabilities	-
Included plans	-
Notes	-

Name	**Cooperative dispatching**
Description	This capability is creating new routes
Goals	Create a new route that is at that moment in time the best option based on the current status of the environment
Processes	Deciding which area is going to be cleaned, deciding which robot is going to take over, making the new route
Protocols	-
Outgoing messages	New route schedule
Incoming messages	-
Internal messages	Area to be cleaned, which robot is going to take over
Percepts	-
Actions	-
Included capabilities	Decide which robot is going to take over, decide which area is going to be cleaned, making new route, schedule new route
Data used: imported	Robot status, system status, forecast
Data internal	New route
Included capabilities	-
Included plans	-
Notes	-

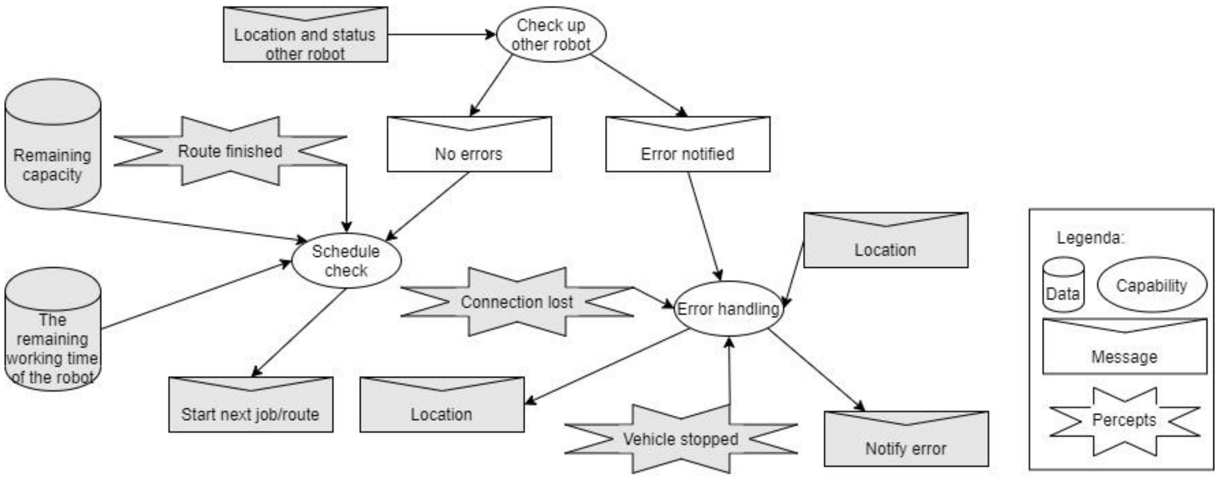
Name	**Location management**
Description	Keeps track of the area around the robot
Goals	To avoid collisions
Processes	Checking if the area is safe if there is another robot near
Protocols	-
Outgoing messages	-
Incoming messages	Status robot closely
Internal messages	Area safe, area unsafe
Percepts	-
Actions	-
Included capabilities	Send error message, Decide if there is a change of collision
Data used: imported	-
Data internal	-
Included capabilities	-
Included plans	-
Notes	-

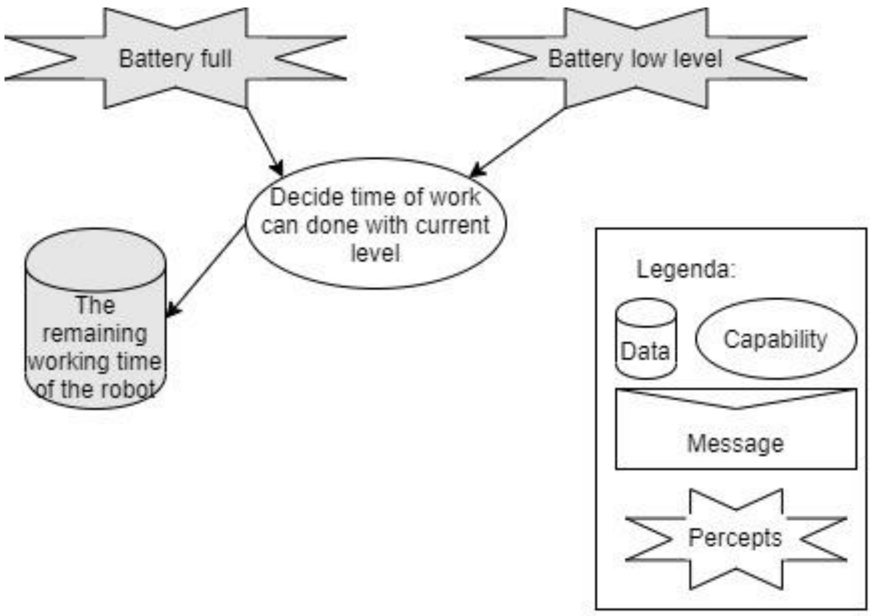
Name	**Capacity management**
Description	This capability keeps track of the remaining capacities
Goals	Avoid full tank during route
Processes	Checking current status capacity
Protocols	-
Outgoing messages	-
Incoming messages	-
Internal messages	-
Percepts	-
Actions	-
Included capabilities	Decide the time of work can be done with the current level
Data used: imported	-
Data internal	-
Included capabilities	-
Included plans	-
Notes	-

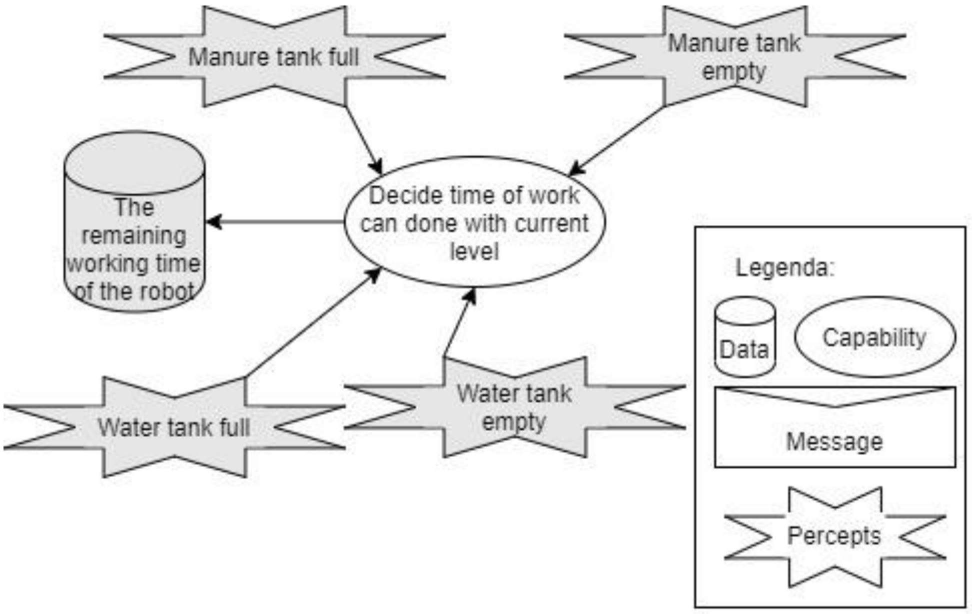
Name	**Battery management**
Description	This capability keeps track of the battery status
Goals	Avoid empty battery
Processes	Checking current status capacity
Protocols	-
Outgoing messages	-
Incoming messages	-
Internal messages	-
Percepts	-
Actions	-
Included capabilities	Decide the time of work can be done with the current level
Data used: imported	-
Data internal	-
Included capabilities	-
Included plans	-
Notes	-

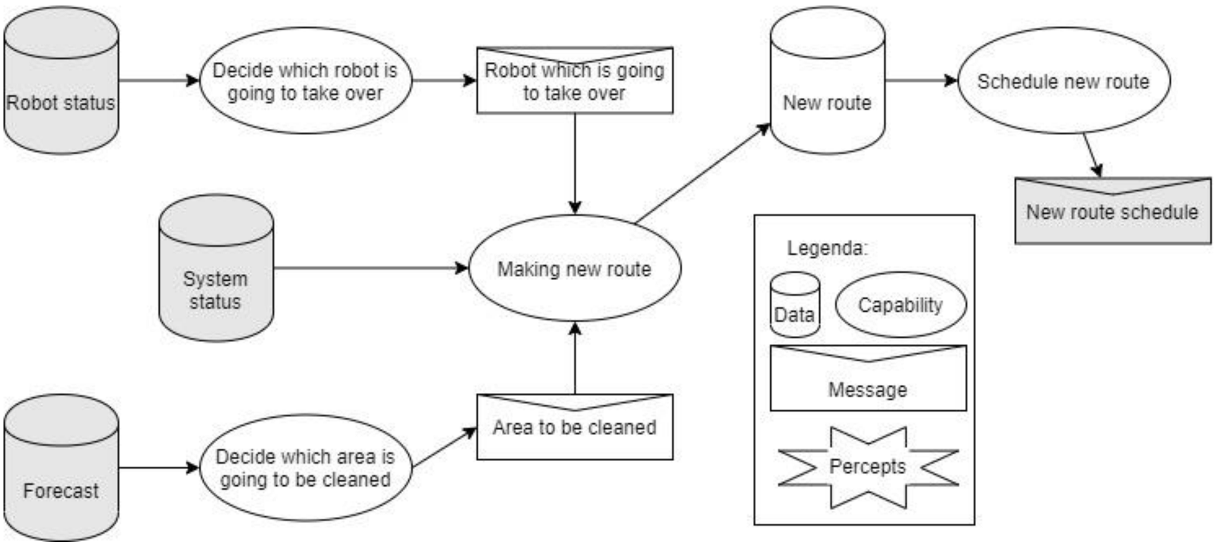
Appendix K. Capability overview diagrams

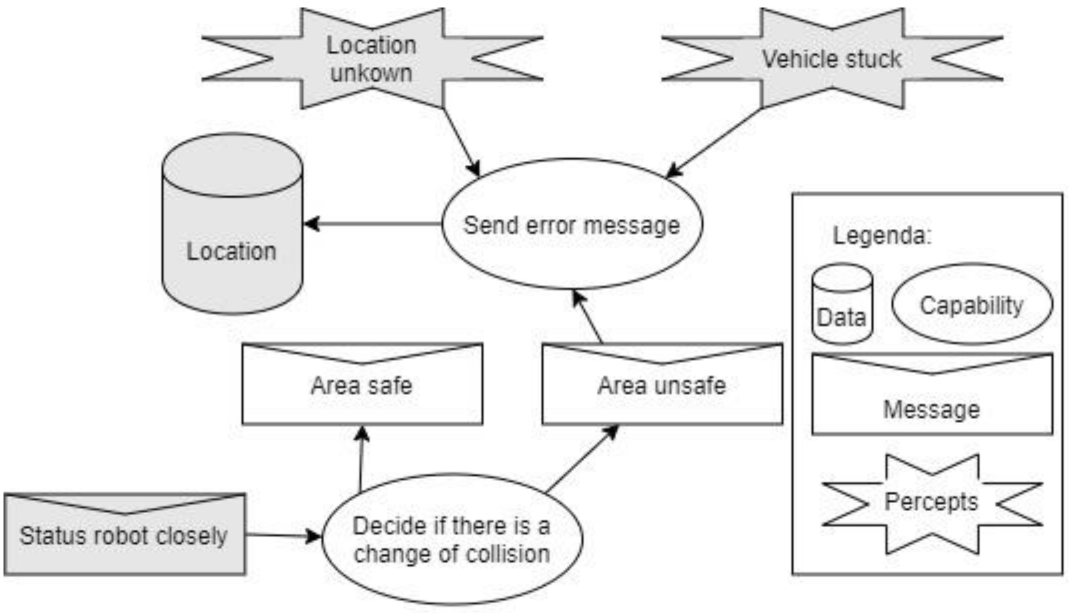
Down here are all the capabilities visualized in capability overview diagrams.

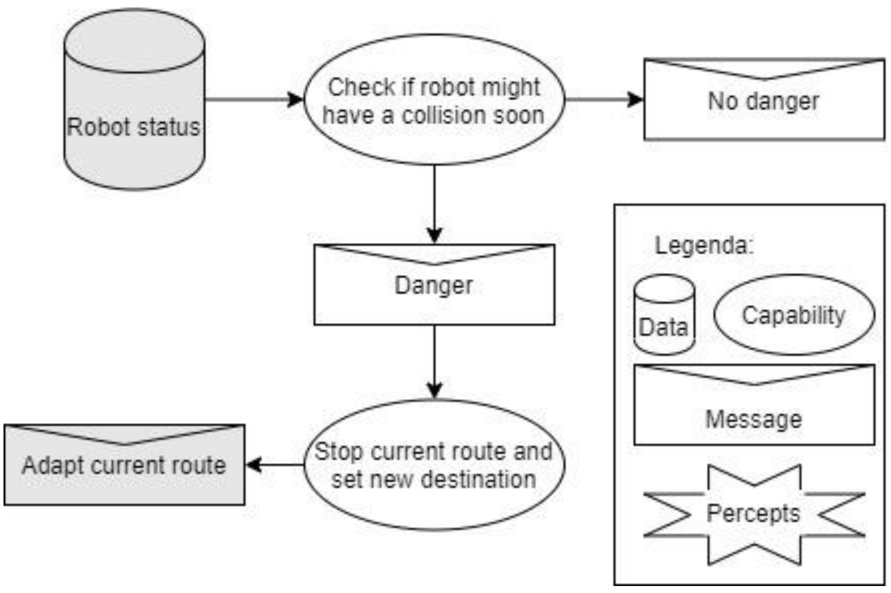



Battery full
Battery low level
Decide time of work can done with current level
The remaining working time of the robot
Legenda:
Data
Capability
Message
Percepts

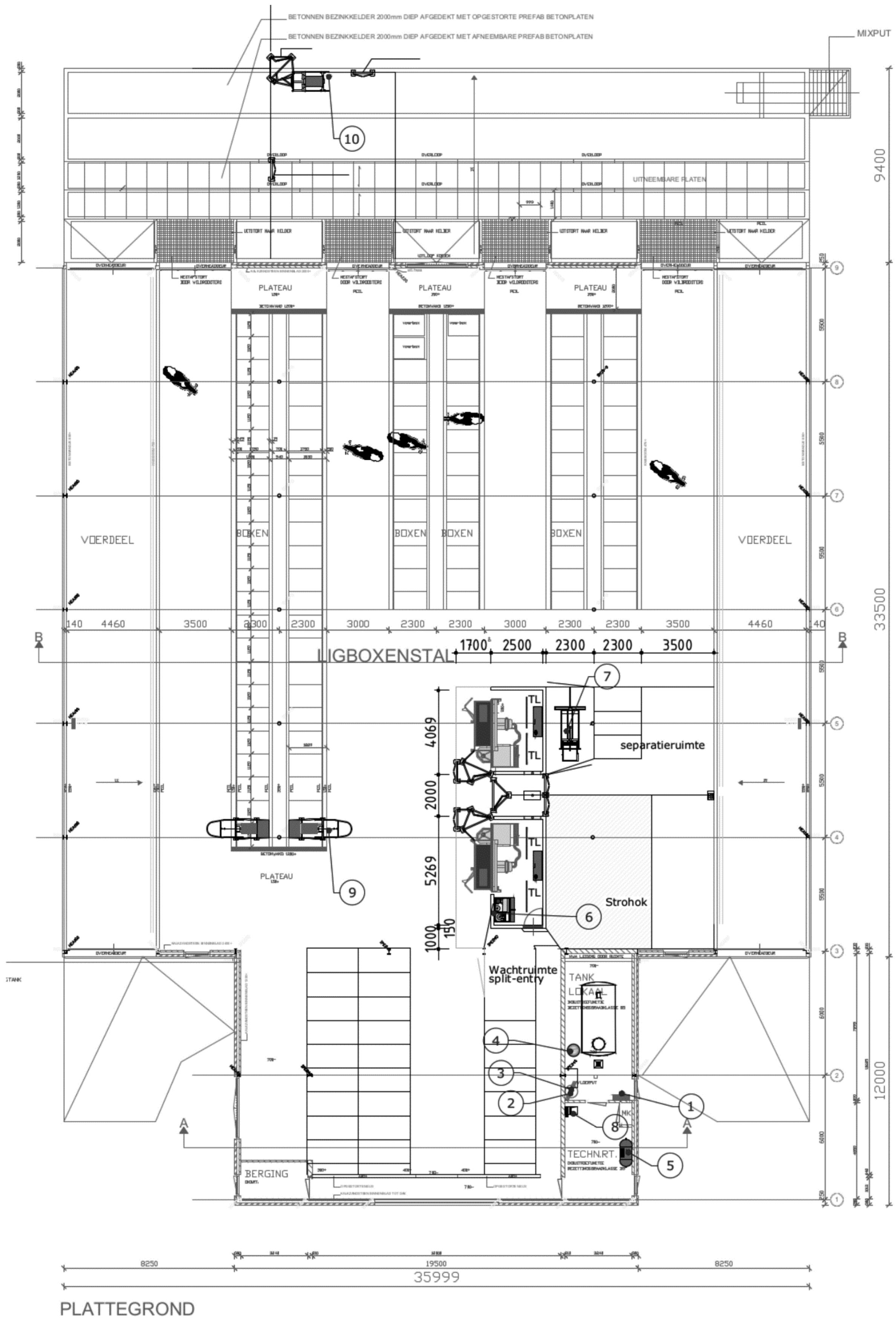

Manure tank full
Manure tank empty
The remaining working time of the robot
Decide time of work can done with current level
Water tank full
Water tank empty
Legenda:
Data
Capability
Message
Percepts


Robot status
Decide which robot is going to take over
Robot which is going to take over
New route
Schedule new route
New route schedule
System status
Making new route
Forecast
Decide which area is going to be cleaned
Area to be cleaned
Legenda:
Data
Capability
Message
Percepts


Location unkown
Vehicle stuck
Location
Send error message
Legenda:
Data
Capability
Message
Percepts
Area safe
Area unsafe
Status robot closely
Decide if there is a change of collision


Robot status
Check if robot might have a collision soon
No danger
Danger
Legenda:
Data
Capability
Message
Percepts
Adapt current route
Stop current route and set new destination

Appendix L. Blueprint testing barn Lely



Appendix M. Scope and level of detail

Component	Include/exclude	Justification
Entities		
Agents	Include	Implement MAS in the simulation model
Manure	Include	Response: quantity of manure in the barn
Dump spots	Include	Response: quantity of manure taken out of the barn
Charger	Include	Response: charging time
AMS	Exclude	The MAS is not interacting in this case with the AMS
Feeding system	Exclude	The MAS is not interacting in this case with the feeding system
Communication points	Include	Experimental factor determines how fast failure messages are sent through the system
Activities		
Driving	Include	Key influence on the process
Docking	Include	Key influence on the process
Cleaning	Include	Key influence on the process
Dumping	Include	Key influence on the process
Spraying water	Exclude	Has influence on the dried manure, which is not taken into account
Route adaption	Include	Key influence on the process, test functionality
Creating new routes	Include	Key influence on the process, experimental factor
Feeding	Exclude	Influence on feeding schedule and process not taken into account due to time constraints
Milking	Exclude	Influence on milking schedule and process not taken into account due to time constraints
Queues		
Dumping station queue	Include	Influences the choice of the routing
Resources		
Cleaning robots	Include	Key resource of the process
Cows	Exclude	We are not looking into robot cow interaction.

Component	**Detail**	**Include/ exclude**	**Justification**
Entities			
Agents	Quantity: depends on agent type and MAS approach	Include	One per system or robot
Manure	Arrival pattern: model input	Include	Increases over time with specific uniform distribution
	Arrival pattern: time-based	Exclude	Excluded due to lack of information and time constraints
	Arrival pattern: event-based	Exclude	Excluded due to lack of information and time constraints
	Other: visualization	Include	Color scale of how much manure is laying on a piece of area
Dump spots	Quantity: depends on barn layout	Include	Number of dump spots is set in the blueprint of the barn
	Attributes: Combined entities	Include	Dump spot can be combined with a charger station
Charger	Quantity: depends on barn layout	Include	Number of chargers is set in the blueprint of the barn, and the number of robots
	Attributes: Combined entities	Include	Charger can be combined with a dump spot
AMS	Other: visualization	Include	AMSs are made visible but with no function
Feeding system	Other: visualization	Include	Feeding spots are made visible but with no function
Communication points	Quantity: experimental factor	Include	Differs per experiment
	Attributes: communication range, experimental factor	Include	Differs per experiment
Activities			
Driving	Routing: fixed schedule	Include	Model input, route dependent
	Routing: backward driving	Exclude	The speed of the cleaning robot is so slow that it does not have an influence
	Routing: backup and go forward	Exclude	The speed of the cleaning robot is so slow that it does not have an influence
	Breakdown/repair: duration night time	Include	Model input following a specific distribution
	Breakdown/repair: duration day time	Include	Model input following a specific distribution
	Breakdown/repair: interval	Include	Model input following a specific distribution
Docking	Quantity: number of times per route	Include	Model input, route dependent
Cleaning	Quantity: % of the time per route	Include	Assume that when the robot is driving, cleaning

	Breakdown	Exclude	Assume that when the robot is driving, cleaning
Dumping	Quantity: number of times per route	Include	Model input, route dependent
	Cycle time: time to dump its manure	Include	Model input
	Breakdown	Exclude	Assume the robot can always dump its manure when it is at the dump spot
Charging	Quantity: at the end of every route	Include	Model input
	Cycle time: time to charge	Include	Model input, route dependent
	Breakdown	Exclude	Assume the robot can always charge its battery when it is at the charger
Spraying water		Exclude	
Route adaption	Quantity: when two robots come across each other	Include	
	Routing: shortest path to end station	Include	Heuristic leads the robot to its end destination (dump spot and/or charger)
Creating new routes	Quantity: when a failure has occurred	Include	When a failure in the environment is occurred and discovered, and a robot is available
	Routing: a new route is created	Include	Heuristic determines a new route based on the area in the barn with the highest urgency
Feeding		Exclude	
Milking		Exclude	
Queues			
Dumping spot queue	Routing: robots are sent to the closest available dump spot	Include	When not driving a standard route, the robots are sent to the closest available dump spot
Resources			
Cleaning robots	Quantity: number of robots working actively in the barn	Include	Model input
	Where required: when cleaning the barn during the routes	Include	The cleaning robots are the resources that are performing the activities
	Other: communication range to other robots	Include	Experimental factor determines how fast failure messages are sent through the system, differs per experiment
Cows		Exclude	

Appendix N. Model description

The model starts with the method Init. This method makes sure that different methods are started in the correct sequence. The first method is called InitGrid. This method creates a new grid of markers on the floor with the correct data whenever it is called. The second method is InitTables. This method initializes all the tables at the start of the simulation. The third one is the ShitGenerator, and this method makes sure that the manure is created on the floor and increases over time. How fast the manure on every marker increases is divided over three categories, and every marker knows which category it is. This method is called a self-execute in every 60 seconds of simulation time.

The fourth method is InitStartPosition. This method sets the destination of every AGV to its start position when the simulation starts. Next to that, it initializes the data the AGV's are carrying. The fifth method is CreateSectors. This method divides the markers over different sectors in the barn. These sectors are predefined based on x and y coordinates. The method CheckDumpSpots checks if the dump spots for manure are reachable for the AGV's. If dump spots are reachable or not are kept in a table. The method CheckSectors is similar to the CheckDumpSpots method but does the check for the sector instead of the dump spot. If a sector can be visited or not is also kept in a table. A sector and dump spots are put on a no-go area if an AGV has an error in that area. When an AGV has an error, it puts all markers in a specific range on no-go markers. This range is because of the location error the AGV has in the actual situation. If one of those markers falls in a sector or the range around a dump spot, it is put on no go.

The method WriteShit collects one of the main KPIs, the average quantity of manure over all the markers per time unit. This data is logged in a table. The method HeatMap makes a heatmap of this KPI. This heatmap indicates how the manure is distributed in the barn at a specific moment in time.

At the beginning of every simulation, the last four methods called in the Init method are responsible for starting the AGV's and the communication in the system. All those four methods are called with a self-execute in once the Init activates them. The method CheckSchedule has as a basic functionality to let the AGV's start their new route at the correct time according to the schedule. This starting is done by the method NextRoute. Also, this method checks if there are any failures in the system, if they are already known and if the system has responded to the failure. This data is kept track of in a table. If a failure has been resolved, the method resets these tables back to normal. If a new failure is discovered in the system, then the method calls another method, BlockErrorArea. And if there is a failure in the system where it is not responded to on jet, and there is an AGV available to take over the job of the AGV with a failure, then the method calls the method NewRoute to create a new route.

The GetPosition method updates the position of each AGV and, with that, the data of the marker where the AGV currently is. Depending on which marker the AGV is, the AGV can also do different actions like dumping manure and charging. If two robots are getting too close to each other, the method RouteAdaption is called to find a new route for one of the two AGV's.

The method CheckUpOtherRobots is continuously trying to communicate with the other AGV's in the system. When it is possible to communicate with each other, they hand over all information about the other AGV's in the system. In that way, failure messages may be quicker communicated through the system. And when an AGV comes across an AGV with an error, the method calls the methods BlockErrorArea and the method RouteAdaption.

The last method, which is called the Init, is the method Communicate. This method is similar to the CheckUpOtherRobots, but the AGV's are communicating with the central system in this method.

The method RouteAdaption is called when two AGV's are getting too close to each other. This situation can happen when an AGV comes across an AGV on his route with a failure. The procedure in this method is that the current route that is driven is stopped, and a new end destination is set. This end destination can be a dump spot or the charger depending on where the route is stopped. The route to this end destination is made by driving point to point. When it arrives at a new point, the AGV searches in a specific range for the next best point to go. This procedure continues until the AGV has gone along all end destinations and is again at its charger.

The method BlockErrorArea is called when there is a new failure is discovered in the system. When a new failure is discovered, the location of the AGV with a failure is also known with a particular uncertainty. To cover this uncertainty, the method blocks an area around the known location. The method calls the method Checksectors at the end to block the sectors overlapping with the blocked area.

The method NewRoute is creating new routes for the AGV's when there is a failure in the system. This creating is done by getting the 10 most urgent markers to visit based on the last time of visit. The sector with the most of those 10 markers is the sector to clean if it is reachable. There is also an option to exclude some of the sectors to be picked. These sectors are the smaller sectors between the bigger lanes.

The next step is to decide which AGV is going to adapt its route. It can be only one option in the use case, but the code is made generic, so it is implementable for cases with multiple AGV's. The decision is made by looking at which AGV has its charging station the closest to the sector that has to be cleaned. This node is the starting node of the route. This node is checked which nodes can directly be visited from the first node only going over one arc. The decision on which node to go to is based on the euclidean distance from that node to one of the two nodes connected to the arc that must be visited. This procedure is repeated until the arc that has to be visited is reached. In the decision-making, some rules are built to prevent the AGV from not unnecessarily driving over arcs and that the heuristic gets not stuck in a loop. When the arc that has to be visited is added to the route, the whole procedure starts again until the end destination, a dump spot of the charger of the robot, is reached. When it is decided which nodes are in the route and sequence, the route is sent to the AGV. This sequence is a sequence of markers instead of nodes. The next step is to calculate the needed battery level for the route. In the end, the method AdaptSchedule is called to put the new route in the planning.

The method AdaptSchedule is called when there is a new route that has to be driven. The method first blocks the standard schedule of the AGV that is going to drive the new route. Then the method puts the new route in the schedule with all needed data.

The method NextRoute, placed on the AGV itself, takes care of starting and driving the correct route at the right moment. The method starts with getting the array of markers that forms the route. Then the method lets the AGV drive the whole route. In the end, there is a check if the route was a takeover route and if the next route is a takeover route. Based on this, the values in the schedule are reset or not.

The method WriteData, which is also placed on the AGV itself, is logging all relevant data of the driven route as soon as the AGV has returned to its charger.

Appendix O. Route analysis

Robot	Route	Starttijd	Eindtijd	Rijtijd	Start dump	Eind dump	Start dump2	Eind dump2	Dumptijd1	Dumptijd2	Totaal Dumptijd	Effectieve rijtijd
2	-			00:00:00					00:00:00	00:00:00	00:00:00	00:00:00
2	-			00:00:00					00:00:00	00:00:00	00:00:00	00:00:00
2	manually	17:23:26	17:27:37	00:04:11					00:00:00	00:00:00	00:00:00	00:04:11
1	r1r1	01:15:03	01:36:23	00:21:20	01:28:32	01:29:32			00:01:00	00:00:00	00:01:00	00:20:20
1	r1r1	03:15:03	03:35:39	00:20:36	03:28:07	03:29:07			00:01:00	00:00:00	00:01:00	00:19:36
1	r1r1	05:15:02	05:35:55	00:20:53	05:28:08	05:29:08			00:01:00	00:00:00	00:01:00	00:19:53
1	r1r1	07:15:02	07:35:57	00:20:55	07:28:03	07:29:03			00:01:00	00:00:00	00:01:00	00:19:55
1	r1r1	09:15:02	09:36:26	00:21:24	09:28:28	09:29:28			00:01:00	00:00:00	00:01:00	00:20:24
1	r1r1	11:15:02	11:35:54	00:20:52	11:28:02	11:29:02			00:01:00	00:00:00	00:01:00	00:19:52
1	r1r1	13:15:03	13:36:28	00:21:25	13:28:35	13:29:36			00:01:01	00:00:00	00:01:01	00:20:24
1	r1r1	15:15:03	15:35:24	00:20:21	15:28:35	15:29:35			00:01:00	00:00:00	00:01:00	00:19:21
1	r1r1	17:15:03	17:35:54	00:20:51	17:28:05	17:29:05			00:01:00	00:00:00	00:01:00	00:19:51
1	r1r1	19:15:03	19:36:25	00:21:22	19:28:36	19:29:35			00:00:59	00:00:00	00:00:59	00:20:23
1	r1r1	21:15:03	21:36:25	00:21:22	21:28:34	21:29:34			00:01:00	00:00:00	00:01:00	00:20:22
1	r1r1	23:15:00	23:35:58	00:20:58	23:28:07	23:29:07			00:01:00	00:00:00	00:01:00	00:19:58
1	r1r2	00:00:02	00:17:08	00:17:06	00:10:14	00:11:14			00:01:00	00:00:00	00:01:00	00:16:06
1	r1r2	02:00:03	02:17:17	00:17:14	02:10:12	02:11:13			00:01:01	00:00:00	00:01:01	00:16:13
1	r1r2	04:00:02	04:17:13	00:17:11	04:10:09	04:11:09			00:01:00	00:00:00	00:01:00	00:16:11
1	r1r2	06:00:03	06:17:06	00:17:03	06:10:08	06:11:08			00:01:00	00:00:00	00:01:00	00:16:03
1	r1r2	08:00:02	08:17:16	00:17:14	08:10:14	08:11:14			00:01:00	00:00:00	00:01:00	00:16:14
1	r1r2	10:00:05	10:16:59	00:16:54	10:10:06	10:11:06			00:01:00	00:00:00	00:01:00	00:15:54
1	r1r2	12:00:02	12:17:05	00:17:03	12:10:05	12:11:05			00:01:00	00:00:00	00:01:00	00:16:03
1	r1r2	14:00:03	14:17:12	00:17:09	14:10:09	14:11:10			00:01:01	00:00:00	00:01:01	00:16:08
1	r1r2	16:00:03	16:17:06	00:17:03	16:10:13	16:11:13			00:01:00	00:00:00	00:01:00	00:16:03
1	r1r2	18:00:03	18:17:05	00:17:02	18:10:07	18:11:08			00:01:01	00:00:00	00:01:01	00:16:01
1	r1r2	20:00:03	20:17:10	00:17:07	20:10:12	20:11:12			00:01:00	00:00:00	00:01:00	00:16:07

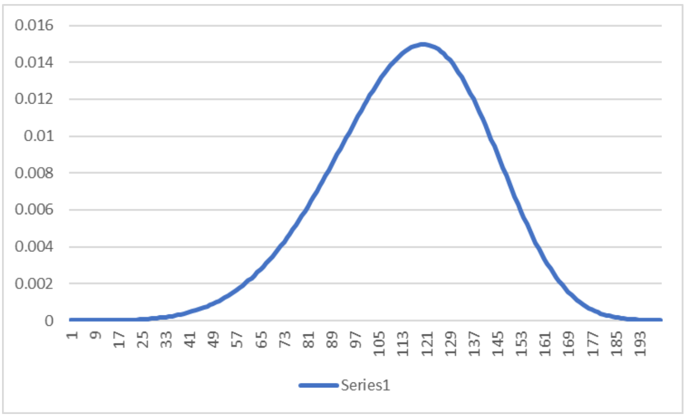
1	r1r2	22:00:03	22:17:05	00:17:02	22:10:06	22:11:06			00:01:00	00:00:00	00:01:00	00:16:02
1	r1r3	02:30:03	02:45:47	00:15:44	02:41:01	02:42:01			00:01:00	00:00:00	00:01:00	00:14:44
1	r1r3	06:30:03	06:46:10	00:16:07	06:41:25	06:42:25			00:01:00	00:00:00	00:01:00	00:15:07
1	r1r3	10:30:03	10:46:19	00:16:16	10:41:33	10:42:33			00:01:00	00:00:00	00:01:00	00:15:16
1	r1r3	14:30:03	14:45:24	00:15:21	14:40:37	14:41:37			00:01:00	00:00:00	00:01:00	00:14:21
1	r1r3	18:30:03	18:45:45	00:15:42	18:41:01	18:42:01			00:01:00	00:00:00	00:01:00	00:14:42
1	r1r3	22:30:03	22:45:19	00:15:16	22:40:34	22:41:34			00:01:00	00:00:00	00:01:00	00:14:16
1	r1r4	04:30:03	04:56:21	00:26:18	04:46:50	04:47:50			00:01:00	00:00:00	00:01:00	00:25:18
1	r1r4	08:30:02	08:56:15	00:26:13	08:47:12	08:48:12			00:01:00	00:00:00	00:01:00	00:25:13
1	r1r4	12:30:03	12:56:20	00:26:17	12:46:53	12:47:53			00:01:00	00:00:00	00:01:00	00:25:17
1	r1r4	16:30:02	16:56:23	00:26:21	16:46:54	16:47:54			00:01:00	00:00:00	00:01:00	00:25:21
1	r1r4	20:30:03	20:55:53	00:25:50	20:46:51	20:47:51			00:01:00	00:00:00	00:01:00	00:24:50
2	r2r1	03:00:02	03:12:08	00:12:06	03:06:16	03:06:46			00:00:30	00:00:00	00:00:30	00:11:36
2	r2r1	07:00:02	07:12:10	00:12:08	07:06:19	07:06:49			00:00:30	00:00:00	00:00:30	00:11:38
2	r2r1	11:00:02	11:12:10	00:12:08	11:06:17	11:06:48			00:00:31	00:00:00	00:00:31	00:11:37
2	r2r1	15:00:02	15:44:14	00:44:12	15:06:58	15:07:28			00:00:30	00:00:00	00:00:30	00:43:42
2	r2r1	19:00:02	19:12:13	00:12:11	19:06:18	19:06:48			00:00:30	00:00:00	00:00:30	00:11:41
2	r2r1	23:00:02	23:12:09	00:12:07	23:06:18	23:06:48			00:00:30	00:00:00	00:00:30	00:11:37
2	r2r2	00:00:02	00:34:58	00:34:56	00:11:00	00:12:00	00:27:44	00:28:13	00:01:00	00:00:29	00:01:29	00:33:27
2	r2r2	02:00:02	02:35:03	00:35:01	02:11:01	02:12:01	02:27:43	02:28:13	00:01:00	00:00:30	00:01:30	00:33:31
2	r2r2	04:00:02	04:34:56	00:34:54	04:11:01	04:12:01	04:27:53	04:28:23	00:01:00	00:00:30	00:01:30	00:33:24
2	r2r2	08:00:02	08:35:32	00:35:30	08:11:26	08:12:26	08:28:12	08:28:42	00:01:00	00:00:30	00:01:30	00:34:00
2	r2r2	10:00:02	10:35:09	00:35:07	10:11:02	10:12:02	10:27:50	10:28:20	00:01:00	00:00:30	00:01:30	00:33:37
2	r2r2	12:00:02	12:36:55	00:36:53	12:11:13	12:12:13	12:28:43	12:29:13	00:01:00	00:00:30	00:01:30	00:35:23
2	r2r2	14:00:02	14:35:52	00:35:50	14:10:51	14:11:51	14:27:49	14:28:19	00:01:00	00:00:30	00:01:30	00:34:20
2	r2r2	18:00:02	18:34:59	00:34:57	18:11:02	18:12:02	18:27:44	18:28:14	00:01:00	00:00:30	00:01:30	00:33:27
2	r2r2	20:00:03	20:35:06	00:35:03	20:11:01	20:12:01	20:27:45	20:28:15	00:01:00	00:00:30	00:01:30	00:33:33
2	r2r2	22:00:02	22:35:05	00:35:03	22:11:05	22:12:05	22:27:22	22:27:52	00:01:00	00:00:30	00:01:30	00:33:33
2	r2r3	01:00:02	01:27:44	00:27:42	01:15:31	01:16:01			00:00:30	00:00:00	00:00:30	00:27:12

2	r2r3	05:00:02	05:27:51	00:27:49	05:15:30	05:16:00			00:00:30	00:00:00	00:00:30	00:27:19
2	r2r3	09:00:02	09:28:02	00:28:00	09:15:56	09:16:26			00:00:30	00:00:00	00:00:30	00:27:30
2	r2r3	13:00:02	13:37:01	00:36:59	13:18:37	13:19:07			00:00:30	00:00:00	00:00:30	00:36:29
2	r2r3	21:00:02	21:27:45	00:27:43	21:15:35	21:16:05			00:00:30	00:00:00	00:00:30	00:27:13
1				00:00:00					00:00:00	00:00:00	00:00:00	00:00:00
2				00:00:00					00:00:00	00:00:00	00:00:00	00:00:00

Appendix P. Failure distribution

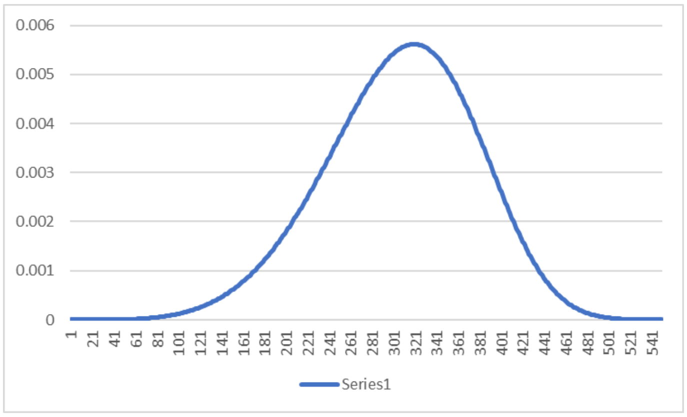
Failure distribution during daytime.

a	30	
b	180	
m	111.8182	
mu	107.4194	
Y		
Alpha	5	
Beta	125.4767	7528.605
Gamma	30	
m	150	



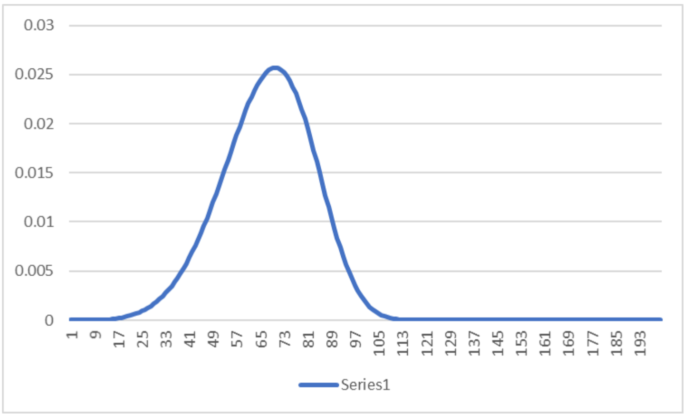
Failure distribution during nighttime.

a	30	
b	480	
m	275.4545	
mu	262.2581	
Y		
Alpha	5	
Beta	334.6047	20076.279
Gamma	30	
m	350	



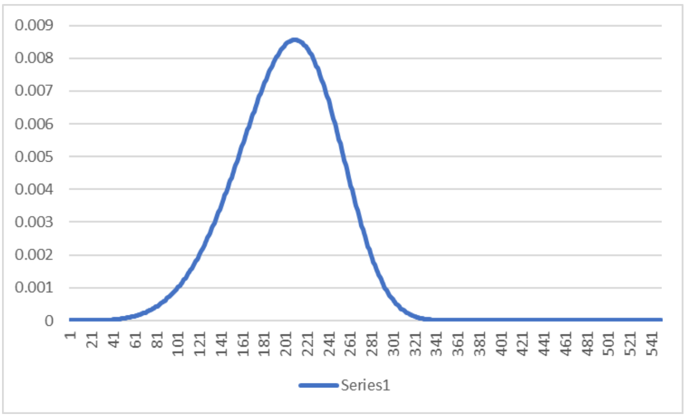
Failure distribution sensitivity analysis during daytime.

a	30	
b	180	
m	111.8182	
mu	107.4194	
Y		
Alpha	5	
Beta	73.19477	4391.686
Gamma	30	
m	100	



Failure distribution sensitivity analysis during nighttime.

a	30	
b	480	
m	275.4545	
mu	262.2581	
Y		
Alpha	5	
Beta	219.5843	13175.058
Gamma	30	
m	240	



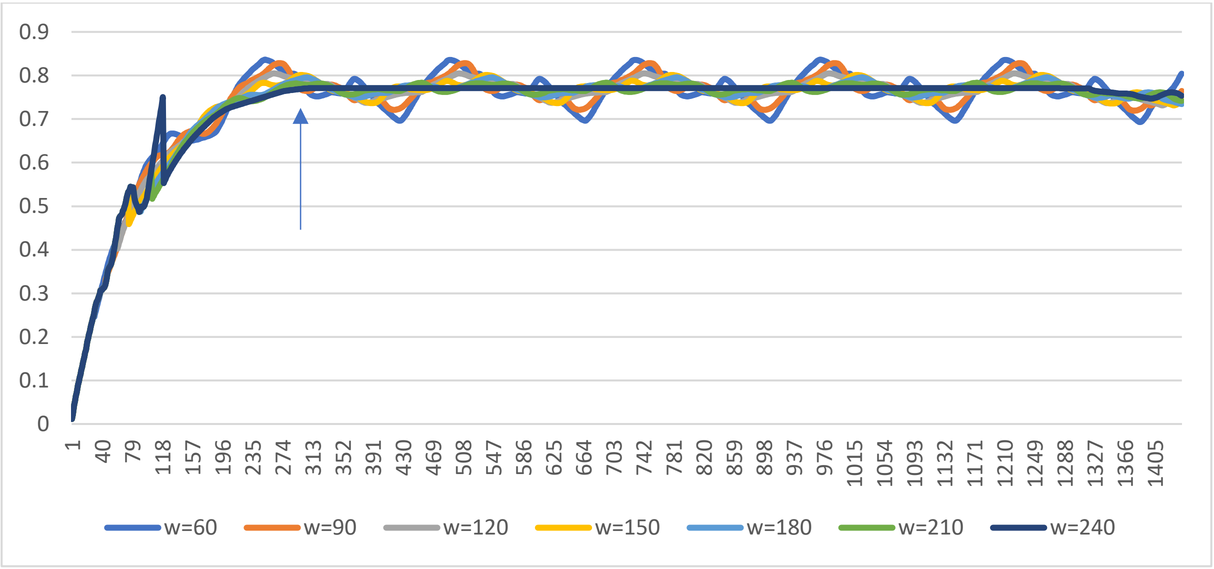
Appendix Q. Validation calculations

Speed without corner reduction							
Lely							
Route	#	Avg Driving Time	Avg Effective Driving Time	Avg Total Effective Driving Time			
r1r1	12	00:21:02	00:20:02	00:20:02			
r1r2	12	00:17:06	00:16:05	00:16:05			
r1r3	6	00:15:44	00:14:44	00:14:44			
r1r4	5	00:26:12	00:25:12	00:25:12			
r2r1	5	00:12:08	00:11:38	00:11:38			
r2r2	10	00:35:19	00:33:50	00:33:50			
r2r3	4	00:27:49	00:27:19	00:27:19			
Simulation							
Route	#	Avg Driving Time	Avg Effective Driving Time	Avg Total Effective Driving Time			
r1r1	12	00:21:57	00:20:54	00:20:54			
r1r2	12	00:16:21	00:15:18	00:15:18			
r1r3	6	00:15:42	00:14:39	00:14:39			
r1r4	6	00:23:00	00:21:57	00:21:57			
r2r1	6	00:14:34	00:14:02	00:14:02			
r2r2	12	00:40:26	00:38:52	00:38:52			
r2r3	6	00:33:17	00:32:46	00:32:46			
Difference (ABS)							
r1r1		00:00:55	00:00:52	00:00:52			
r1r2		00:00:45	00:00:48	00:00:48			
r1r3		00:00:02	00:00:05	00:00:05			
r1r4		00:03:12	00:03:15	00:03:15			
r2r1		00:02:26	00:02:25	00:02:25			

r2r2		00:05:07	00:05:02	00:05:02			
r2r3		00:05:29	00:05:27	00:05:27			
Factor							
r1r1	12	1.04	1.04	1.04	12.52		
r1r2	12	0.96	0.95	0.95	11.41		
r1r3	6	1.00	0.99	0.99	5.97		
r1r4	6	0.88	0.87	0.87	5.23		
r2r1	6	1.20	1.21	1.21	7.24		
r2r2	12	1.14	1.15	1.15	13.79		
r2r3	6	1.20	1.20	1.20	7.20	Factor	Speed
	60				63.3454	1.0557	0.1583

Appendix R. Simulation set-up

Down here are the calculations and diagrams shown used for the simulation set-up.



n	Average	Avg	Var	T-value	Error	Gamma	0.05
1	0.79					Gamma'	0.052632
2	0.81	0.80	0.00	12.7062	0.184941		
3	0.78	0.79	0.00	4.302653	0.052235		
4	0.81	0.80	0.00	3.182446	0.032753		
5	0.79	0.79	0.00	2.776445	0.022731		

n	Average	Avg	Var	T-value	Error	Gamma	0.05
1	0.81					Gamma'	0.052632
2	0.84	0.83	0.00	12.7062	0.18946		
3	0.79	0.82	0.00	4.302653	0.067002		
4	0.80	0.81	0.00	3.182446	0.036802		

5	0.87	0.82	0.00	2.776445	0.044017		
6	0.81	0.82	0.00	2.570582	0.0347		
7	0.82	0.82	0.00	2.446912	0.027934		
8	0.80	0.82	0.00	2.364624	0.024782		
9	0.84	0.82	0.00	2.306004	0.022415		
10	0.81	0.82	0.00	2.262157	0.020064		